UiO : Department of Informatics
University of Oslo

Study of how handovers in mobile broadband affects TCP

Patrick Skevik

Master's Thesis Spring 2015

Study of how handovers in mobile broadband affects TCP

Patrick Skevik

4th February 2015

Abstract

This thesis studies how changing from 4G to 3G in mobile broadband affects an active TCP connection. This has been done by analysis of several tests done with different modes of transportation. Testing has been done with both UDP and TCP with a comparative analysis performed and involves both the client and server side. Results show that TCP has a slower throughput growth after a handover, when compared to UDP, and can suffer from extended periods of inactivity, also referred to as stalling.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

The total number of smartphones sold in 2013 alone, is estimated to be over 950 million [1]. Since 2008 the number of activated android devices is over 900 million alone, with an estimated 1.5 million activations per day. These numbers represent the intense growth of mobile devices. This poses new challenges for mobile broadband.

In 2013 the number of mobile-cellular subscriptions is estimated to be 6 662 million and the number of active mobile broadband subscriptions is estimated to be 1 930 million [2].

As the number of people with online access through devices grow, the number of new online services are growing too. More and more services ranging from online gaming, to video streaming is becoming normal and people are utilizing them even more so from mobile devices.

This increase in usage poses new challenges for mobile broadband providers. Carrier networks must increase their service to stay competitive and to ensure that their network can handle bandwidth heavy services like YouTube and Netflix over mobile broadband.

Digital availability is becoming a not only a new norm, but a requirement. Having access to the internet at all times is becoming more and more common as our daily computer and device usage is increasing. We expect to be able to receive emails, chat messages and notifications through our smartphones, tablets and laptops at any time and any place.

This increase in users is massive and is growing by millions every year, this puts a big strain on the current networks. Mobile broadband demand is increasing and to be able to support all these new devices, the service demand requires a lot of work. Mobile broadband is currently expanding with new technologies in order to meet the demand. Despite this, there are still problems. 4G is the latest standard which attempts to meet the current and future demand, but the deployment is still lagging behind. Networks are still expanding 4G coverage, but for now 3G and 2G have a far better coverage. This means that we still rely

on older standards and technologies like 3G and 2G to cover users with mobile broadband.

This means that new smartphones must support older technologies to stay competitive. It has to be able to use 2G, 3G and 4G which requires independent hardware and each of these generations have incompatible sub-technologies, it can require independent hardware for different technologies within these generations. An example is LTE (Long Term Evolution) and WiMax, both of these are considered 4G technologies, but are based on different architectures and function differently. In order for a smartphone to support both of these it would require independent hardware within the device itself.

One issue users are experiencing with this is the constant change of technology. When a user moves outside of a 4G covered area the device must be able to switch technology, while still maintaining the users active connections. This can occur when a user is travelling by bus, or is walking into a parking house. Depending on what kind of device it can cause significant overhead, leaving a web-site seemingly unresponsive despite the connection being available. This can be quite frustrating, especially if the switch in technology happens frequently. Subways which can travel both above ground and below and a user can go from 4G to 3G to 2G in a matter of minutes when travelling from above ground to below.

1.2 Problem statement

This thesis studies how a handover can affect an active TCP connection. The term handover is overloaded and can have several different meanings. In this thesis the term handover is limited to when a modem changes technology between different mobile broadband generations.

A handover can be the result of a lost signal, when a modem has lost communication with a base station belonging to a specific technology. When this occurs, the modem will attempt to re-establish a connection to an alternative technology. This can happen when a user is connected to 4G, but is moving beyond the coverage area, but is still in an area with 3G coverage.

What happens to the TCP connection is the main study of this thesis, with a goal of attempting to understand any underlying issues, and whether or not there is any potential for improvement in the TCP protocol.

1.3 Thesis structure

This thesis is structured in the following way:

The first chapter introduces this thesis with some motivational reasoning for this problem, followed by the problem statement and a description of the thesis structure. Chapter 2 covers the background of the thesis in order to capture the problem domain. Chapter 3 describes

the methodology of how the tests have been conducted and how the data has been gathered and with what software. Chapter 4 describes the results and presents an analysis of the results. Chapter 5 discusses the results and analysis and will present future work and limitations within the thesis. Chapter 6 contains the conclusion of the thesis.

1.4 Research goal

The main goal of this thesis is to understand how a handover affects an active TCP connection, and will attempt to discover any underlying issues in the protocol with a focus on whether or not it is possible to introduce potential improvements on TCP.

Chapter 2

Background

This chapter will give a background in mobile broadband, describing the different generations before giving an introduction to TCP and UDP.

2.1 Mobile Communication

About every tenth year, a new mobile broadband standard is released. The first mobile communication system was released in USA in October, 1983 [3]. This was an analog system, allowing voices to be transferred over radio waves using a wireless phone.

The release of the second generation mobile communication system was in 1991. This was the release of 2G and was commercially launched in Finland by a company called Radiolinja [4]. This was a big step up from the first generation as this was a fully digital system.

Ten years later, the third generation (3G) was released in 2001, expanding even further on the second generation. It was first released in Japan. During the course of 2002 more carriers around the world started to offer 3G commercially.

Fourth generation, the current generation, was commercially available in 2009 [5]. While it has been debated whether to call it 4G or pre-4G, the technology used is named Long Term Evolution (LTE) and is currently widely adopted as a 4G technology.

2.2 History

This section will introduce the history of mobile broadband and explain some of the technologies which are currently in use today and how they operate.

2.2.1 Second Generation

The advent of the second generation brought a lot of changes to mobile communication. It was a switch from analog to digital. The main benefit was the ability to digitize audio, to convert analog voices into digital voices. This allowed for compression as well as improving security with

the ability to encrypt the digitized communication. With the ability to compress voices, it was now possible to squeeze in more users per cell tower, than the previous system could. This would allow more users to use the system at the same time.

There has been several 2G technologies created and used all over the world. This has lead to incompatibility for users as one mobile device could not be guaranteed to work when a user travelled abroad. Below is a short list of 2G technologies will be listed and some will be briefly expanded upon.

- Interim Standard 95 (IS-95)
- Digital Advanced Mobile Phone System
- Personal Digital Cellular
- Global System for Mobile Communications

Interim Standard 95 (IS-95), was one of the first 2G technologies available. It was developed by Qualcomm and unlike GSM, it utilizes a technology known as Code Division Multiplex Access (CDMA) which is a channel access technology. It was in competition with DAMPS in North America.

Digital Advanced Mobile Phone System (DAMPS) was the digital expansion of the previous 1G Advanced Mobile Phone System (AMPS) system. This was mainly used in the USA and Canada, but has since 2005 been discontinued in favour of GSM. It used a channel access system called Time Division Multiple Access (TDMA) and used an encryption named Cellular Message Encryption Algorithm (CMEA) to encrypt calls.

Global System for Mobile communication

It started out as a single mobile standard set in Europe under the name ”Group Spécial Mobile”, but it was later renamed to ”Global System for Mobile communication” as to appeal to a broader audience.

GSM got started at an informal conference in Paris in 1980 with the goal of exchanging views on mobile radio development and was looking for common ground on how to use the 900 MHz spectrum. In the beginning, most nations had a low expectation as to the adoption of mobile communication. While GSM wasn’t developed until years after this conference, it was here that nations started the talks of using a common standard [6] .

As a digital system, GSM opened the door for new ways to look at mobile communication. It was first released in 1991.

Because this system was digital, it could convert analog voices in to discreet digitized voices, allowing a conversation to be split up into thousands of tiny data packets. This combined with a channel access

technology named Time Division Multiple Access (TDMA), users could now utilize the same frequency without causing interference at the same time.

TDMA allocates time slots for each user and in this time slot, the user has a full bandwidth. This goes for both uplink and downlink communication.

In 2001 a new service was introduced to the GSM architecture named General Packet Radio Service (GPRS). This would allow general packets to be transferred over the GSM system, allowing a user access to the internet over mobile communication. It has since then become a core part of the telecommunication network.

In order to keep up with the increasing demand, in 2003 GPRS was extended with Enhanced Data rate for GSM Evolution (EDGE). It allows for a greater download and upload speed, and is considered to be a 3G technology.

2.2.2 Third Generation

The third generation mobile broadband was set to offer users a higher data transfer rates and greater worldwide compatibility. It began in the 1980s and was in development for 15 years, and was first commercially launched in Japan, 2001 by NTT DoCoMo [7]. Since then it has been widely adopted across the world.

The 3G mobile broadband standard was released in 1998 and a 3G technology is defined as a technology that complies with the International Mobile Telecommunications-2000 (IMT-2000) specifications.

There are two main technologies branded as 3G, while EDGE does comply with the standard, it is mostly regarded as a pre-3G, post-2G technology. These two projects were the result of 3GPP and 3GPP2, two different 3G collaboration projects with one focusing on the upgrade of GSM and the other on the upgrade of IS-95 respectively.

Universal Mobile Telecommunication System (UMTS) is a component of the IMT-2000 standard and defines a mobile cellular system for 3G networks. It specifies a complete network including, but not limited to, UMTS Terrestrial Radio Access Network (UTRAN) and authentication with SIM (Subscriber Identity Module) cards. It is an upgrade from the GSM network and is based on a similar architecture.

With the deployment of HSPA+ enhancement, 3G networks can now reach bandwidth levels comparable with current 4G LTE technology.

CDMA2000 is the other 3G branded technology and was developed and used in North America, China, Japan and others. It is an upgrade to the IS-95 standard and in turn also uses CDMA as a channel access technology [**CDMA2000**].

Technology	Description
W-CDMA	Wideband Code Division Multiple Access uses CDMA as a channel access method, with a pair 5 MH/z channel and has a bandwidth of 384Kbit/s.
HSDPA	High Speed Downlink Packet Access is an extension to UMTS which increases the downlink bandwidth to 21Mbit/s, it is also refereed to as 3.5G or TurboG [8].
HSUPA	High Speed Uplink Packet Access is similar to HSDPA, except it extends the uplink transfer rate up to 5.74 Mbit/s [9].
HSPA+	Evolved High Speed Packet Access enhances WCDMA with speeds comparable to LTE. It increases the bandwidth available for mobile devices significantly, with a downlink up to 42 Mbit/s [10].

Table 2.1: Different air interfaces for UMTS

2.2.3 Fourth Generation

The requirements for the fourth generation mobile broadband technology was published by ITU-R (ITU Radiocommunication sector) in march 2008 and was named International Mobile Telecommunications Advanced (Also known as IMT-Advanced). It defines, among other things, the peak speed for downlink and uplink depending on mobility. For a highly mobile user (Car, train, bus) it must provide 100 Mbit/s and for a low mobility user (Sitting in a coffee shop) up to 1 Gbit/s.

There was originally two competing 4G technologies, Long Term Evolution (LTE) and Mobile WiMax. Mobile WiMax was first commercially available in South Korea, 2007 while LTE was launched in 2009. These two technologies have different architectures and are based on different technologies. While LTE is based on similar architectures as 3G and 2G technologies, WiMax is based on IEEE 802.16 family. These two were competing technologies, but it was LTE that became the widespread and adapted across the world.

Long Term Evolution

LTE is the name given one of the 4G technology candidates. The standard was developed by the 3GPP (3rd Generation Partnership Project) and is based on similar architecture as GSM/UMTS. It uses

more current digital processing methods than the previous technologies and uses different parts of the spectrum [11].

It has been debated as to call LTE a 4G technology or a pre-4G technology as it has been unable to reach the bandwidth requirements of 1Gbit/s for stationary users and 100 Mbit/s for mobile users. An extension to LTE, LTE-Advanced, is currently in process and has been widely tested and should be able to meet the 4G requirements.

2.3 TCP

This section will explain how TCP is structured and how it works with a focus on parts that will be relevant throughout this thesis.

Transmission Control Protocol (TCP) is a transport layer protocol. It is used to deliver streams of packets. It is one of the core protocols of the Internet Protocol Suite. It was proposed in 1974 when its first specification was published under RFC 675 [12].

Since then it has changed a lot with new added features and mechanisms. Among these flow control and congestion control.

TCP is a connection-oriented transport layer protocol, its function is to transport data from one end to another and to ensure that the packets was received and acknowledged by the receiving end. Built in the protocol is error-checking, by validating that the header and data is correctly received as well as prevent the network from congesting by sending too much data over the network.

It has become a cornerstone of the internet and is one of the most used transport protocols today.

It is used in a variety of services and areas to ensure proper communication. Below is a sort list of areas where it is commonly used.

- File transfer
  When transferring a file from one machine to another, it is imperative that the file is not corrupted. It should be possible to transfer files over slow and even partially faulty networks, without corrupting the file it self.

- Video streaming
  Online video services like Netflix and YouTube use TCP with a buffer cache at the client. This is to ensure a live feed with minimal interference.

- World Wide Web
  To access a web page a user must download a web-file, which makes browsing web similar to file transfer.

- Chatting
  When sending a message to someone else, it is necessary to send the whole message and not just parts of it.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Source port	Destination port
Sequence number	
Acknowledgement number (if ACK set)	
TCP Header length / Reserved / NS CWR ECE URG ACK PSH RST SYN FIN	Window Size
Checksum	Urgent pointer (if URG set)
Options (if TCP Header length > 5, padded at end with "0" bytes if necessary)	

Table 2.2: TCP Header [13]

The remaining parts of this section will describe the structure of TCP's header, and how flow control and congestion avoidance works.

2.3.1 Structure

The TCP header structure is displayed in figure 2.2 and an explanation of the fields will follow promptly.

Source, Destination ports (32 bits)

The ports defined in the first 32 bits represents the local end points of the connection.

Sequence Number (32 bits)

Sequence numbers have two usage areas. If it is set to '1', then it represents the initial sequence number. Otherwise this number will represent the current sequence in packets being transfer.

It is increase by adding the length of the previous packets with the initial value, and if it increases to the point of the maximum value, it will roll back, starting from the bottom and grow up.

Acknowledgement Number (32 bits)

If the acknowledgement number is set, then the value represents the next expected sequence number. This also acknowledges all prior bytes, which means that it will acknowledge all packets with a sequence number lower to this number.

TCP Header length (4 bits)

This field describes the length of the TCP header, and is required as the options field is of variable length. It is used in order to determine where the header ends and where the data begins.

Reserved (3 bits)

These bits are reserved for the future.

Flags (9 bits)

Each flag is represented with 1 bit (Either set, or not set).

- NS - ECN-nonce concealment protection
- CWR - Congestion Window Reduction lets the user know that the server has reduced the congestion window.
- ECE - Explicit Congestion Notification is set to let the sender know to slow down when the client receives congestion notification from the network.
- URG - If this is set to 1, then it indicates that the Urgent pointer field is used.
- ACK - When this is set, it indicates that the Acknowledgement number is valid. Otherwise the ack number will be ignored.
- PSH - This is used to push the data to the receiving application.
- RST - Reset the connection.
- SYN - This bit is used to establish a connection, this will be explained later.
- FIN - No more data from sender.

Window size (16 bits)

This window size field lets the sender know how many bytes the receiver can buffer.

It should be noted that this field can also be zero, which is a way of letting the server know that the receiver is unable to buffer any data packets at this time. This is known as a Zero Window packet and it is used to temporarily block the connection.

During this time the server will reply with keep-alive packets, simple packets sent to the receiver to ensure that the receiver is still connected.

Once a receiver can buffer packets, it will retransmit a window-update packet with the new window size.

Checksum (16 bits)

Contains the checksum of the header and data, used for error checking.

Urgent pointer (16 bits)

The urgent pointer is used to point to an offset in the sequence numbers in which urgent data is found. This is rarely used.

Options (Variable 0-320 bits, divisible by 32)

Options have up to three fields, which is dependent on what kind of option. The tree fields are: Option-Kind (1 byte), Option-Length (1 byte) and Option-Data (Variable).

Option-Kind is mandatory and can be a No option kind which is used for padding. If option-kind is zero, then it represents the End-of-Options option. These options can be maximum segment size, window scaling and selective acknowledgement.

Padding

Padding might be required to ensure that the TCP header ends and data begins on a 32 bit boundary. This is composed of zeroes.

The TCP header has remained largely unchanged for years, with the exception of a few new additional flags relating to congestion. These new flags are the ”NS”, ”CWR” and ”ECE”. The last two were added in 2001[14], while the last addition was added in 2003[15].

The protocol behind TCP is described by an internal state diagram as shown in figure 2.1 . This describes how both the client and server acts when starting and closing a connection. During the established state, the sender and receiver exchanges packets.

During the packet exchange, the client will have sent an ACK after the handover. This will contain the next expected packet with a sequence number equal to the ACK packets Acknowledgement field.

Some implementations maintain a delayed ACKing, in which the receiver will wait until it has received more packets before ACKing, this works as a way of combining ACK packets and prevents the receiver from sending too many ACK packets. The receiver can delay ACKing up to 500 ms.

When a client is connecting to a server, it follows a handshake protocol, which can be seen in the diagram mentioned. It starts with the client sending a packet with the SYN flag set to 1, and the ACK flag set to 0. When the ACK flag is zero, it means that the server can ignore the Acknowledgement field.

The server responds with a packet containing the flags SYN set to 1 and ACK set to 1.

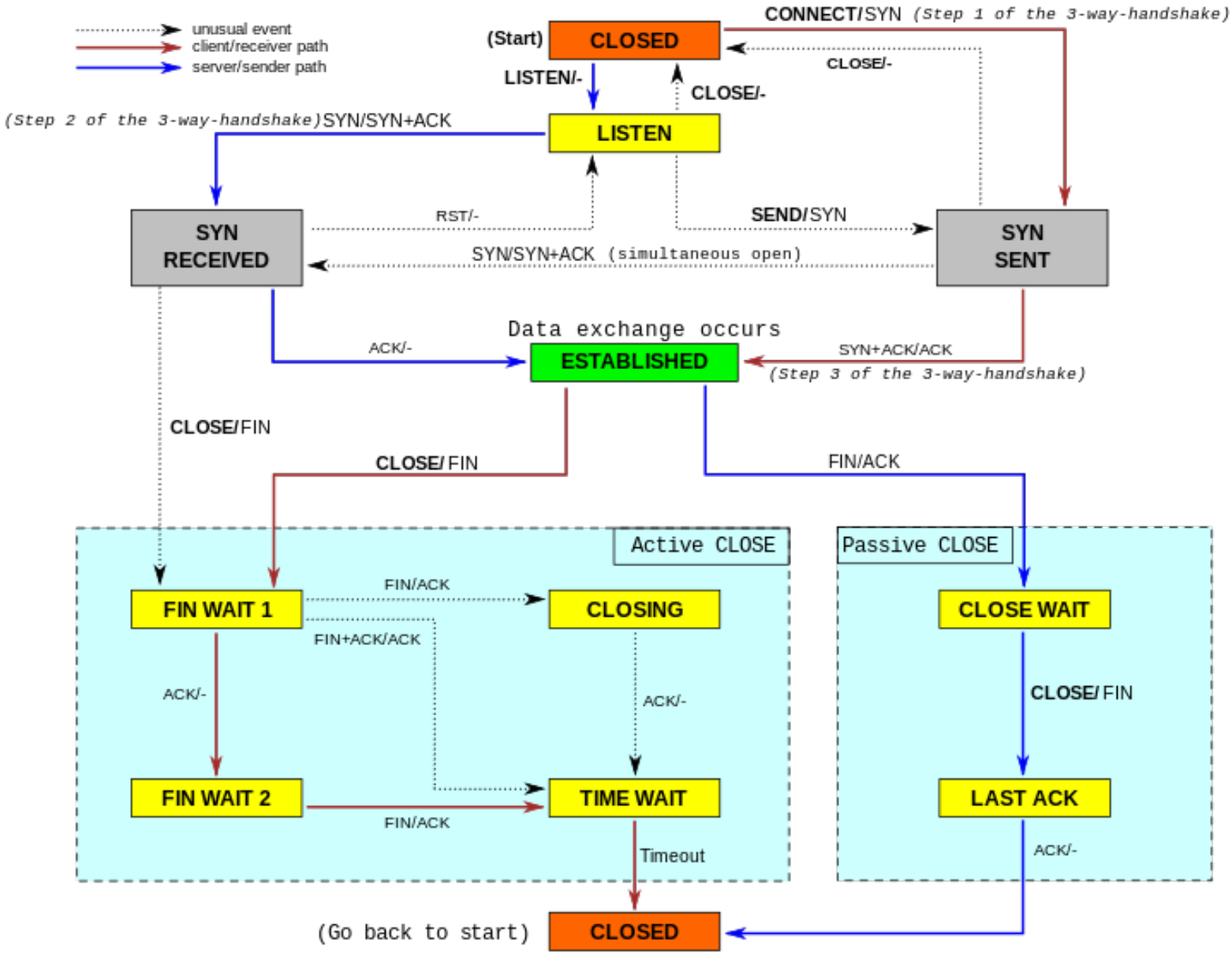

Figure 2.1: TCP state diagram, showing the different internal TCP protocol states [16].

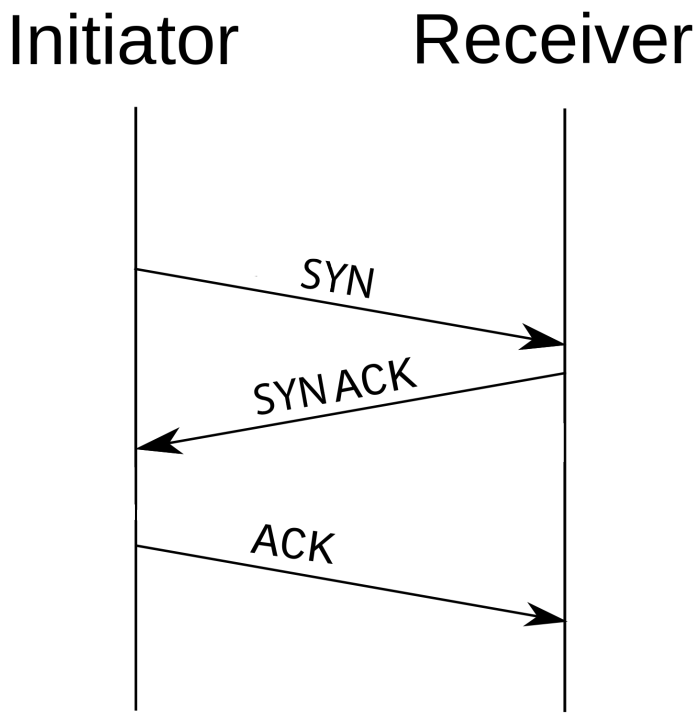

Figure 2.2: How a TCP connection is established.

This 3-way handshake can be seen in figure 2.2.

Closing the connection can be done in several ways, but a proper close can be seen in the diagram. A client maintains an active close method, while the server keeps a passive close flow. This requires a 4-way handshake and is initiated by the client. This can be seen in figure 2.3.

Flow control is handled by the protocol to ensure that both parties are receiving as much data as they can handle. This is important to prevent flooding of the connection, where one end receives more data than they can handle which subsequently leads to packet loss and an unstable connection.

TCP uses a concept called Sliding Window and an example can be seen in figure 2.4 .

This example shows how windows slide as data packets with increasing sequence numbers are received. It shows how a window slides as it receives data.

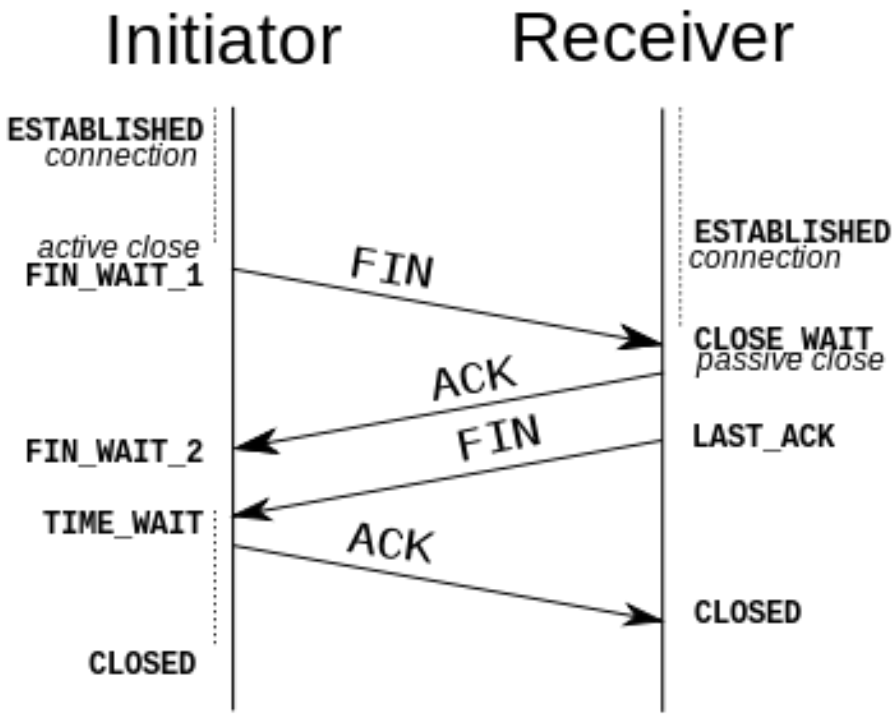

Figure 2.3: How a TCP connection is closed [17].

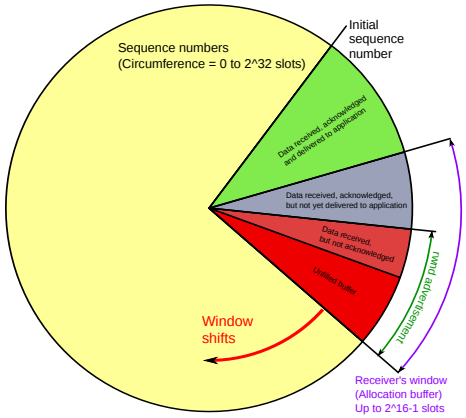

Figure 2.4: A visual example of sliding window [18].

Packet Loss and duplicate ACKs

When there is packet loss in a network, the receiver will receive packets out-of-order. This means that the packets sequence number is higher than the expected number. Once this happens, the receiver will send a duplicate ACK, this means it will retransmit the last sent ACK.

A duplicate ACK can contain selective acknowledgement (SACK) [19]. SACK is an option that must be enabled when establishing the

connection and it is set in the options field. SACK will store the sequence number of packets that are out-of-order to let the sender know which packets are missing, and where the holes in the transmission is.

Duplicate ACKs are used by TCP to determine packet loss, as it is a useful indication that the network is suffering from congestion. It should be noted that while duplicate ACKs will be used when the receiver is experiencing packet loss, packet reordering will also trigger this.

When the sender sends packets with sequence numbers 11, 12, 13 and 14, they can be reordered during the transfer between the sender and receiver, and the receiver can get them in a different order example: 14, 13, 12, 11. This will result in duplicate acknowledgements which can be interpreted as packet loss.

Packet loss can be caused by many issues, as each packet can visit several routers on their way to the receiver. As a path is not predetermined, but rather sent in a best-effort, there is no way to predict the path a packet will take.

2.3.2 Congestion Control

Congestion control is an integral part of TCP. It is a set of algorithms that is used to ensure that the connection remains stable, and does not suffer from ”Congestion collapse”.

It consists of four algorithms[20]:

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery

Slow start and congestion avoidance is used when there is no packet loss and which of the two algorithms used depends on two internal variables. These are Slow Start Threshold (ssthresh) and Congestion Window (cwnd).

Fast Retransmit and Fast Recovery are used when the receiver is suffering from packet loss. Not both are implemented in every congestion algorithm.

Congestion window is used to limit the number of packets being sent in order to prevent congestion in the network. It takes precedence of the receiving window of the receiver if the sender knows that the network is unable to handle the advertised window. SSThresh is the Slow Start Threshold.

Slow start is used when cwnd is less than ssthresh and congestion avoidance is used when cwnd is greater than ssthresh. When the value of cwnd is equal to ssthresh, it is up to the implementation to determine which one to use.

The differences between the four algorithms is outlined below.

Slow Start

This algorithm is used when the congestion window is lower than the slow start threshold (ssthresh) and is usually active in the beginning of a connection, or after a connection has been repaired after packet loss.

It will increase the congestion window until it is equal or exceeds the ssthresh value for each incoming ACK packet. Then it will change to congestion avoidance.

Congestion Avoidance

This algorithm is used when the sender is attempting to prevent congestion in the network. For each completed RTT the congestion window will increase by 1, until duplicate ACKs are received in which congestion is detected.

Fast Retransmit

When the sender receives 3 duplicate ACKs, it will change the ssthresh value and start to retransmit the missing packet(s) without waiting for a retransmission timeout.

The new value of ssthresh depends on which congestion control algorithm is used. It is usually set to half the congestion window size, but different algorithms have different approaches.

Once the sender has sent the missing packets, it will change to Fast Recovery.

Fast Recovery

This algorithm will continue to handle the transmission of new data packets, until the congestion window is equal to, or greater than the ssthresh value. In which case, slow start will take over.

Congestion collapse can occur when the amount of data being transferred is greater than the networks capabilities. This can result in routers filling up their memories and starting to drop packets. This can also be the cause when there are spurious retransmissions, resulting in low useful throughput.

In order to limit the number of packets in flight, a congestion window is used. This will limit the rate of packets being sent in order to avoid congestion. Example:

Congestion Algorithm	Description
Tahoe	Tahoe implements Slow start, Congestion Avoidance and Fast Retransmit, but does not have a Fast Recovery algorithm. Instead of switching to fast recovery after Fast Retransmit, it simply changes to Slow Start.
Reno	Reno is an extension of Tahoe except with Fast Recovery implemented.
New Reno	New Reno is an improved version of Reno, with an improved retransmission and fast recovery algorithm. It also improves on duplicate acknowledgement packets among others [21].
Cubic	CUBIC is an extension to the BIC algorithm, it uses a cubic function to calculate the window growth and is a less aggressive than its predecessor [22]. CUBIC is the default congestion control for Linux since kernel 2.6.19 to 3.1.

Table 2.3: List of Congestion control algorithms

If a receiver is advertising a receiving window of 64KB, but the sender knows that the network can only handle 32KB, then the sender will only send 32KB.[13, p.590]

There are several varieties of this algorithm, each with different methods of acting when congestion is discovered. A description of different congestion control avoidance algorithms is listen in list 2.3. There are many other algorithms and this is not a complete list.

2.3.3 Retransmission Timer

When the sender sends a packet, a timer is set and when the timer expires, the packet is retransmitted. This is called a retransmission timeout, or RTO.

The length of this timer is standardized in RFC 6298 [23]: "Computing TCP's Retransmission Timer" and a summary of this will follow.

The algorithm states that until a round-trip time has been completed, the timer must be set to 1.

Once a round-trip time (RTT) has been completed, the RTO shall be set to the smoothed round-trip time (SRTT) added with the maximum value of either the clock granularity, or the 4*(RTT / 2).

After that, the values will be modified over time, based on new

Port #	Protocol
21	FTP
22	SSH
25	SMTP
53	DNS
80	HTTP
443	HTTPS

Figure 2.5: A list of common ports and protocols they belong to.

complete RTTs.

When a packets timer expires, it is retransmitted and restarted with a double length. This will continue until the timer has reached 60 seconds, and will then continue until the connection times out.

This is also known as ”Exponential Backoff” as the doubling of the timer length grows exponentially.

2.3.4 Linux Implementation of TCP

The implementation of TCP in Linux does not follow the standard completely. It deviates in a few areas and does not handle congestion control in the way it is defined.

Internally, Linux uses TCP states, these are listed in table 2.4 [24]. How TCP reacts to packet loss and congestion depends on which state it is.

These states will then control which algorithm is used, whether it is the slow start algorithm, congestion avoidance, fast recovery or fast retransmit.

Linux also redefines a few predefined values, like the maximum retransmission timer. In the reference it is defined to be 60 seconds, while Linux uses 120 seconds.

These changes will affect the testing, as the server runs Linux.

2.3.5 Ports

A port is a communication endpoints used by applications or processes to allow a communication with the operating system. It is also commonly used in networking as a single machine can run several network applications at the same time. This requires a means to differentiate which application is supposed to receive which packets, in which the port field used in transport protocols come in handy.

Ports are described using 16-bits which allow for 65,536 different possible ports. A short list of the most common ports can be seen in 2.5

Only a few of these are standardized.

- Open
  This is the normal state in which the TCP sender follows the fast path of execution optimized for the common case in processing incoming acknowledgements. When an acknowledgement arrives, the sender increases the congestion window according to either slow start or congestion avoidance, depending on whether the congestion window is smaller or larger than the slow start threshold, respectively.

- Recovery
  When receiving three duplicate ACKs, TCP will enter this state. During this state, the congestion window is reduced by one for every second incoming ACK. The window reduction ends when the congestion window size is equal to ssthresh. A retransmission time-out can also interrupt this state.

- Disorder
  When the sender detects duplicate ACKs or selective acknowledgements, it moves to this state. In this state the congestion window is not adjusted, but each incoming packet triggers transmission of a new packet.

- Loss
  When an RTO expires, the sender enters this state. All outstanding packets are marked as lost and the congestion window is set to one and the sender starts to increase the congestion window using slow start. This state cannot be interrupted by any other state, thus the sender exits to the Open state only after all data outstanding when the Loss state began have successfully been acknowledged.

- CWR
  When receiving a congestion notification, the Linux sender does not reduce the congestion window at once, but by one segment for every second incoming ACK until the window size is halved. When the sender is in process of reducing the congestion window size and it does not have outstanding retransmissions, it is in CWR (Congestion Window Reduced) state. CWR state can be interrupted by Recovery or Loss states.

Table 2.4: List describing the internal Linux TCP states [24]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Source port	Destination port
Length (Header + Data)	Checksum
Data ...	

Table 2.5: UDP Header

2.4 UDP

This section will give a short explanation of what UDP is.

User Datagram Protocol was formally defined in 1980 as RFC 768[25] and is part of the Internet protocol suite. It is a connectionless-oriented protocol. This means that UDP does not establish a connection between the sender and receiver. Instead it just transports data from one end, to the other.

UDP is considered to be an unreliable transport protocol as it has no means of knowing if a sent packet is received or if it was dropped in the network, yet it is still one of the most used transport protocols today.

The UDP header contains 4 fields, each 2 bytes long. It contains a source port, destination port, length and checksum of the data and header. Using this checksum, it is possible for the receiver to check if there are any errors in the header or data. A figure of the protocol can be seen in table 2.5 .

2.5 Internet Protocol

This section will describe what IP is and how it relates to TCP and UDP.

Internet Protocol (IP) is defined in RFC 791 [26], which defines a method of transferring data from one machine to another. IP is used combined with either TCP or UDP (But not limited to either, it can support other transport protocols).

There are currently two versions of IP in use. These are outlined below.

2.5.1 Internet Protocol version 4

IPv4 is the fourth development of IP and is used to route most packets on the internet. It uses 4 bytes to represent a machines address giving it an address space of 2^{32}. This means that IPv4 can represent about 4.2 billions addresses.

Because of the limitations in the address space, it has been said that the internet is running out of addresses. This has led to a new version being created with a far higher number of available addresses.

2.5.2 Internet Protocol version 6

IPv6 uses 16 bytes to represent a machine, giving it an address space of 2^{128}.

While the IPv6 adoption rate has been slow, it has also been steady and is also increasing.

2.6 Summary

This chapter has presented a background around mobile broadband and present some of the history around the different generations as well as presented two transportation protocols and a short description of IP.

The next chapter will present the methodology for this thesis.

Part II

Analysis

Chapter 3

Methodology

This chapter covers the methodology used in this thesis in order to study the problem stated in section 1.2 , this include covering the hardware and software used to test, collect and analyse data as well as generate graphs.

3.1 Testing Setup

This section describes the setup of the testing. This will include the hardware specifications used during the test and a list of the software used during and after the testing phase.

3.1.1 Hardware

The list of hardware used in this thesis:

- **Laptop**
  The laptop used is a Toshiba Satellite Z830 with Ubuntu 13.10 saucy with kernel version 3.11.0-26 and processor Core i5.

- **ZTE MF821D**
  This is a 4G usb modem which supports LTE, UMTS and GSM. LTE Downlink data rate is up to 40Mb/s and HSPA+ downlink is 42Mb/s with HSDPA up to 21.6 Mb/s[27].

- **ZTE MF823**
  This is a 4G usb modem which supports LTE, UMTS and GSM. LTE downlink data rate is up to 100Mb/s and HSPA+ is 42Mb/s and HSDPA up to 21.6 Mb/s[28].

- **Server**
  When conducting tests a server at Simula Research Laboratory has been used. It has been utilized for both TCP and UDP testing, and network traffic has been captured on both this machine

and the laptop during tests along with internal TCP states and congestion window.

3.1.2 Software

In order to gather relevant data several programs have been used in conjunction to first gather, extract, analyse and then to plot figures to look for trends and other potential valuable parts. Below is a list of programs used to do this. In this list a short description of what the tool does follows.

Among these are network capturing tools, programs to extract data and statistics from modems used, network capture analysis tools and others.

- **TCPDump**
  This is a network capturing tool. It captures network traffic and has been used during tests to capture the packet traffic on both the client and server. It creates a pcap (Packet Capture) file which can be used in a variety of different software. [29]

- **Wireshark**
  Wireshark is an interactive packet capture and analysis tool. It can capture network packets and analyse the result. For this thesis, it has only been used as an analytical tool and all the dumps analysed has been imported from the TCPDump tool. [30]

- **Multi client**
  This is a local DHCP client tool, created by Kristian Evensen and has been used in conjunction with the QMI Dialer tool in order to retrieve modem statistics from the ZTE MF-821D modem. [31]

- **QMI Dialer**
  This is a Qualcomm mobile modem communication tool. It has been used to gather data about the modem, as it has no other means of extracting current signal strength, technology and so on.[32]

- **Bandwidth Estimator**
  This is a bandwidth estimation tool similar to Iperf. This tools was only used for UDP testing.[33]

- **TCP Probe**
  This is a module that records TCP connections when it receives ACK packets[34]. It was slightly modified to include the internal TCP state and ports used for the stream in the output. The

changes is based on a modified version of TCP probe which can be found here: https://bitbucket.org/bendikro/tcp_probe_rdb

The changes can be found in the appendix.

- **GNUPlot**
  Gnuplot is a tool to generate plots and figures. It has been used in order to better represent the data captured during this thesis[35].

- **Wget**
  GNU Wget is a non-interactive downloader of files from the Web. It supports HTTP, HTTPS and FTP and can also use HTTP proxies[36].

- **PACKet Graph**
  PACKet graph is a simple graphing system written in Javascript that is based on D3.js to show when packets were sent from the server and received at the client.

- **PHCC**
  Post Handover Connection Checker (PHCC) is a tool created and used in this thesis for several tests. It works by having the sender send a stream of data, with a high throughput. When the client stops receiving TCP packets it starts to send UDP packets to the server, which the server should then respond do.

  In effect, once the client stops receiving TCP packets it sends UDP pings to the server.

  This has been used to determine when the client can receive TCP packets after a handover.

  It is available in the appendix.

Only a few of these have been used on the server which are limited to TCPDump and TCP probe. TCPDump was used to capture the network traffic and TCP probe was used to capture relevant congestion window data and internal TCP data.

3.1.3 TCP

The goal of the tests is to get a better understanding of how TCP is affected by a handover from both the user and servers perspective.

These tests are also used in an attempt to answer a few questions:

- How long does it take before the server and client starts exchanging data after a handover?

- Can TCP be improved before and after a handover?

As stated earlier, when a client is undergoing a handover it will be unresponsive from the servers perspective. With these tests it should be possible to explain what happens when this is the case and any potential problems this will cause should become apparent.

It will also be interesting to observe how a server behaves as the user is experiencing a degrading connection, when the signal falls, and become unresponsive.

3.1.4 UDP

To better understand how a handover affects a users connection, UDP testing has been conducted. One of the goals of UDP testing is to will reveal more about the state of the network post-handover.

In these tests the client is a fully passive receiver, meaning it will only receive data packets without sending any kind of data back to the server. The program used to generate UDP traffic will transmit data for a predetermined amount of time. This has been limited to 120 seconds with 20 Mbit/s.

Ideally UDP testing should reveal a lower limit for when TCP can potentially start to receive data, as the client should start to receive data as soon as possible post handover. This could be useful to determine if there is any issues with TCP.

- Does TCP and UDP behave differently when the client is experiencing a handover both before and after?
- Is there a difference in throughput growth between TCP and UDP?

3.2 Testing methodology

This section will go into the testing methodology and explain some of the reasoning behind the tests done.

Different methods of testing have been conducted for this thesis. Tests have been done several methods, walking, driving in a car and bus. Each of these attempt to simulate a real life situation where a user might experience a handover as a result from moving outside of a 4G coverage area.

Below will be a more detailed explanation of each method.

- **Walking**
  In order to get more real life test walking up and down a parking house has been used to gather data as the modems have moved out of 4G range, while still in a 3G range. This test has been used to capture the handover between 4G and 3G, which can be used to determine how this movement can affect active TCP connections.

- **Driving by car**
  Another real-life situation test has been done by driving around in a card, specifically by driving through tunnels. In tunnels a modem might lose connection to 4G while still having decent 3G coverage. This test will capture the handover of a moving vehicle driving through a tunnel.

- **Handover by bus**
  Testing while on board a bus has been another method. The bus route drove from a 4G covered area into an area with only 3G and 2G.

These tests represent real-life situations which should capture most of the potential handovers a user might experience while travelling. These should uncover potential problems and by studying each of these case individually and combined it should be possible to discover independent problems and problems they have in common.

3.2.1 Limitations

There are several limitations in this thesis regarding what kind of tests, what kind of hardware used and network carrier limitations. These will be outlined below.

This thesis is limited to only observing how a 4G to 3G handover affects a users connection. This is done out of time constraints, as well as the problem of finding areas without 4G to do this testing on, these is also no 3G to 4G handovers captured. This is because as of this thesis there are no handover mechanism from 3G to 4G.

While some 3G to 2G and 2G to 3G handovers have been captured, there have not been significant enough of these to warrant an analysis.

The modems used have been limited to ZTE MF821D and ZTE MF823 due to them being the resources available and their popularity in Norway.

The network carriers are limited to Netcom and Telenor as they both maintain their own LTE networks in Norway.

These tests are also limited to only testing for the effect a handover has on the downlink by downloading a big file or sending a stream of packets. There has been no testing of the uplink of how a handover might affect a user uploading data over mobile broadband. This was done as a user is more likely to download, or stream data rather than upload data over mobile broadband.

3.3 Data collection

This section will explain how data was collected, which data got collected and from where the data was collected.

3.3.1 Network traffic

Capturing network traffic has been essential in this thesis. This has been done using the tcpdump tool, which will output the packets in a pcap format. This allows the network dump to be analysed and imported by a variety of tools. However, only Wireshark has been used for capture analysis in this thesis.

This tool has been used on both the client and server to capture data from both the receiver and sender.

3.3.2 Modems

Modem information was gathered by using two separate programs as both modems had separate interfaces for extracting the relevant connection information. One was through a software that enabled the Linux machine to access it, the other was through a REST API running on the modem.

ZTE MF821D

To gather data from the ZTE MF821D modem, the qmi-dialer was modified to dump the relevant data to file. This software was used because the modem did not have any Linux driver support which would allow access to the connection data.

The data dumped was as follows:

```
...
[13:47:27 15/7/2014 qmi_nas.c:405]: LTE. RSSI -29 dBm RSRQ -7 dB RSRP -53 SNR 28 # bars 4
[13:47:27 15/7/2014 qmi_nas.c:431]: Received RF_BAND_INFO_RECV_RESP
[13:47:27 15/7/2014 qmi_nas.c:447]: 1405424847 8
[13:47:27 15/7/2014 qmi_nas.c:453]: Technology 8 Band 145
...
```

During a handover the modem will return the following:

```
...
[14:9:53 27/5/2014 qmi_wds.c:93]: Could not connect, no service
[14:9:53 27/5/2014 qmi_wds.c:95]: 1401192593 0
...
```

ZTE MF823

This modems runs Linux internally and maintains a web server which displays the connection status. It uses an internal REST API which retrieves the modem status which includes what kind of technology the modem is connected to, signal strength and others. It fetches the data using a single endpoint, which accepts a set of commands to retrieve the data, depending on which data.

This endpoint is available at: "http://192.168.32.1/goform/goform_get_cmd_process" and takes several parameters. The ones used to gather data for this thesis have been limited to the following:

```
http://192.168.32.1/goform/goform_get_cmd_process?cmd=signalbar,network_type,network_provider,ppp_status,
modem_main_state,EX_SSID1,ex_wifi_status,EX_wifi_profile,m_ssid_enable,total_tx_bytes,total_rx_bytes,
total_time,sms_unread_num,sms_received_flag,sts_received_flag,RadioOff,simcard_roam,lan_ipaddr,
spn_display_flag,plmn_display_flag,spn_name_data,spn_b1_flag,spn_b2_flag,station_mac,battery_charging,
battery_vol_percent,battery_pers,pin_status,loginfo,realtime_tx_bytes,realtime_rx_bytes,realtime_time,
realtime_tx_thrpt,realtime_rx_thrpt,monthly_rx_bytes,monthly_tx_bytes,monthly_time,date_month,
data_volume_limit_switch,data_volume_limit_size,data_volume_alert_percent,data_volume_limit_unit
```

```
{
        profile_name: "",
        m_profile_name: "NetCom",
        network_provider: "NetCom",
        wan_ipaddr: "",
        lte_rssi: "-92",
        lac_code: "af1",
        cell_id: "20d1b1e",
        lte_band: "7",
        lte_rsrp: "-129",
        lte_rsrq: "-18",
        prefer_dns_auto: "",
        standby_dns_auto: "",
        enodeb_id: "",
        tx_power: "22",
        dns_mode: "auto",
        prefer_dns_manual: "",
        standby_dns_manual: "",
        rssi: "-73",
        lte_pci: "179",
        ipv6_prefer_dns_auto: "",
        ipv6_standby_dns_auto: "",
        ipv6_wan_ipaddr: "3930:0000:0000:0000:3113:a001:7636:1ff0",
        ipv6_pdp_type: "IP",
        ipv6_prefer_dns_manual: "",
        ipv6_standby_dns_manual: "",
        pdp_type: "IP",
        ipv6_dns_mode: ""
}
```


Figure 3.1: JSON response when using the first set of commands


```
&multi_data=1&sms_received_flag_flag=0&sts_received_flag_flag=0&isTest=false&_=1406883925204

http://192.168.32.1/goform/goform_get_cmd_process?isTest=false&cmd=profile_name,m_profile_name,network_provider,
wan_ipaddr,lte_rssi,lac_code,cell_id,lte_band,lte_rsrp,lte_rsrq,prefer_dns_auto,standby_dns_auto,enodeb_id,
tx_power,dns_mode,prefer_dns_manual,standby_dns_manual,rssi,lte_pci,ipv6_prefer_dns_auto,
ipv6_standby_dns_auto,ipv6_wan_ipaddr,ipv6_pdp_type,ipv6_prefer_dns_manual,ipv6_standby_dns_manual,
pdp_type,ipv6_dns_mode&multi_data=1&_=1406886569981
```


These were gathered from the web interface which displayed relevant data regarding current technology the modem is connected to and the signal quality of the connection. During a handover it reports nothing.

The relevant data was extracted using a script and was executed once every second. The response is encoded in JSON.

Typical JSON responses can be seen in figures 3.1 and 3.2. These are the JSON objects returned by the API.

3.3.3 Internal TCP data

To gather internal TCP data from the server a kernel module named TCP probe was used. This module outputs data to a file located in '/proc/net/tcpprobe'.

Example of this data is as follows:


```
872.654722133 128.39.36.93:8080 46.15.111.172:1032 44 1288792659 1287878179 708 707 1968128 78 14592 4 TCP_CA_Loss
872.654762058 128.39.36.93:8080 46.15.111.172:1032 44 1288793947 1287879467 708 707 1968128 79 14592 4 TCP_CA_Loss
872.654782380 128.39.36.93:8080 46.15.111.172:1032 44 1288795235 1287880755 708 707 1968128 79 14592 4 TCP_CA_Loss
872.654804603 128.39.36.93:8080 46.15.111.172:1032 44 1288796523 1287882043 708 707 1968128 79 14592 4 TCP_CA_Loss
```


The modifications to TCP probe adds the last two parts in the output lines and ports. The last string shows the internal TCP state, in this case it is in the loss state, while the second to last number represents the internal integer value for this state.

Each line starts with a time since the module was loaded followed by the local IP and remote IP. The next values are length of the data, next

```
{
        signalbar: "5",
        network_type: "DC-HSPA+",
        network_provider: "NetCom",
        ppp_status: "ppp_connected",
        modem_main_state: "modem_init_complete",
        EX_SSID1: "0001softbank",
        ex_wifi_status: "",
        EX_wifi_profile: "",
        m_ssid_enable0 ",
        total_tx_bytes: "",
        total_rx_bytes: "",
        total_time: "",
        sms_unread_num: "0",
        sms_received_flag: "",
        sts_received_flag: "",
        RadioOff: "0",
        simcard_roam: "Home",
        lan_ipaddr: "192.168.32.1",
        spn_display_flag: "0",
        plmn_display_flag: "1",
        spn_name_data: "",
        spn_b1_flag: "0",
        spn_b2_flag: "0",
        station_mac: "",
        battery_charging: "",
        battery_vol_percent: "",
        battery_pers: "",
        pin_status: "0",
        loginfo: "ok",
        realtime_tx_bytes: "",
        realtime_rx_bytes: "",
        realtime_time: "",
        realtime_tx_thrpt: "",
        realtime_rx_thrpt: "",
        monthly_rx_bytes: "",
        monthly_tx_bytes: "",
        monthly_time: "",
        date_month: "0",
        data_volume_limit_switch: "0",
        data_volume_limit_size: "",
        data_volume_alert_percent: "",
        data_volume_limit_unit: "
}
```


Figure 3.2: JSON response when using the second set of commands

sequence to send, first byte we want an ACK for, congestion window (cwnd), slow start threshold (ssthresh), sender window, smooth rtt, receiving window and the internal TCP state (Repeated in a string).

Ports was also added in order to separate different streams, as the server would run the TCP probe for several hours while doing the testing.

3.4 Data extraction

This section will give a brief description of the different tools used to extract the data gathered mentioned in the previous section.

3.4.1 tshark

To extract data of interest from the TCP dumps tshark has been used. This tools works by using the terminal with a specific set of parameters and it will output filtered data.

In this thesis it has been used to extract throughput data for further analysis and creating data sets for graphs.

3.4.2 Probe

A small python script was used to divide the connections from the TCP probe output into separate files. This was done to make it easier to analyse the different streams.

3.4.3 Modem data

Two simple scripts were written to extract the technology and signal quality data from both modems. Each modem required a different script as they output data in two different ways. One was written in Python while the other was written in Javascript and executed using NodeJS.

These would filter out the technology used along with a timestamp and the signal quality. Because the modem ZTE MF823 used an internal API, it outputted technology data differently if it was connected to 4G or 3G.

3.5 Summary

This chapter has presented the methodology for this thesis along with a list of the software used during testing and hardware with a short description of each tool.

It has presented the data that was gathered and how data was extracted. The next chapter will present the results and analysis of the testing.

Chapter 4

Result and Analysis

This chapter presents the results and analysis of the tests done for this thesis. The first section will give an overview of the main problem followed by a more in depth analysis.

In these tests the client has been a passive receiver, which means that the server has sent data to the client, but the client does not transmit data back with the exception of ACK packets.

4.1 Overview

This section will give an overview of how a handover affects an active connection when the user is downloading data by observing the throughput graph. In this section several throughput graphs from three tests are used as an example to show how a handover can affect a connection.

When a user is connected to 4G and is moving away from the area with coverage, the modem will lose signal strength. As he moves further away outside the coverage area and loses signal almost entirely, the modem will start to search for a better connection. It will attempt to find a more suited mobile broadband technology to ensure a good connection for the user.

If the modem is able to connect to another technology, either 3G or 2G, it will initiate a handover in the network. During this time the user has temporarily lost access to the network and internet which means also any current active connection

However, it should be noted that while the user loses access to the internet and any connections he may have active, they are not closed. From the servers perspective the user is still connected, but will remain unresponsive during this time.

The next part will show what the effects of the beginning of a handover has on an active connection.

4.1.1 Before Handover

Before the handover occurs, the signal quality will degrade. This effect can be observed by a simple throughput graph as the throughput is falling until the modem changes technology and the user stops receiving and sending data packets.

We can observe the signal degradation in the following figures: 4.1 , 4.2 and 4.3 . These figures shows the throughput graph of three different tests, as the signal is decreasing before a handover occurs.

In figure 4.1 the graph shows a declining throughput until about 20 seconds into the test. Then it falls quickly in a few seconds before halting completely. During this time, the signal is decreasing while the surrounding noise is increasing. At around 20 seconds the signal starts dropping rapidly and the connection collapses as the modem starts to change technology. There was, however, no packet loss before the handover occurs in this capture.

Unlike figure 4.1 , in figure 4.2 there is no slow decrease in the throughput of the connection. Instead the modem loses signal quickly and within 4 seconds the connection has collapsed and communication has grinded to a halt, while the modem is changing technology. Both of these graphs show a common trend.

It should be noted that as the signal is declining, once it reaches a certain threshold the communication becomes more bursty as can be seen in the previous mentioned figure at the end. A spike of packets are received just before the modem starts the handover.

The third graph, as seen in 4.3 , shows a combination of the two. After about 20 seconds the signal starts to decrease, as does the throughput. 10 seconds later the connection starts behaving erratic as the signal quality falls quickly. The connection becomes more bursty and the client receives data in minor bursts. The modem starts to change technology and the connection becomes silent during this transition.

What these graphs show is the different ways a falling signal quality will affect a connection. Because of the nature behind mobile broadband and the many different ways a signal can be disrupted, it is hard to anticipate when a handover will occur. There are indicators, but there is no clear sign from the TCP connection it self, that the signal quality is going to decrease, as it can happen within a few seconds.

A more detailed analysis of these captures will be described in section 4.3.1 which will go deeper into the TCP parts describing how a client and server experiences the handover and how it affects the internal TCP state for the server.

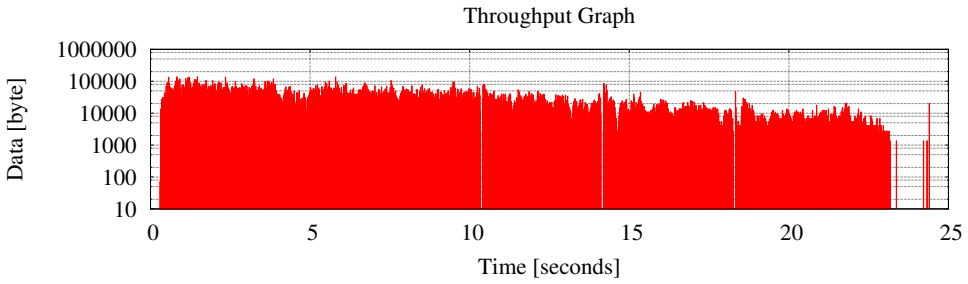

Figure 4.1: TCP Throughput graph using ZTE MF823 with Netcom with a sample rate of 10 milliseconds.

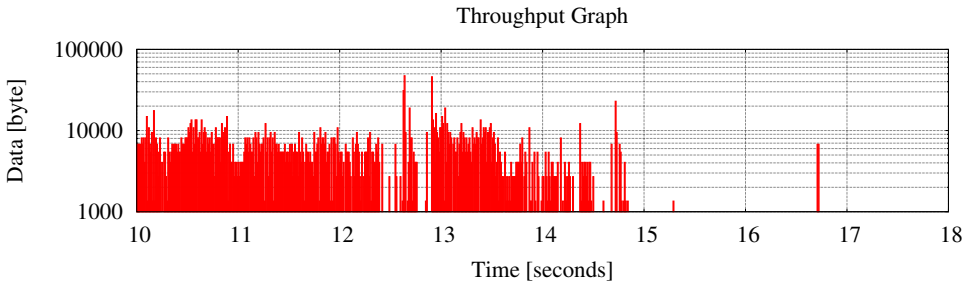

Figure 4.2: TCP Throughput graph using ZTE MF823 with Netcom with a sample rate of 10 milliseconds.

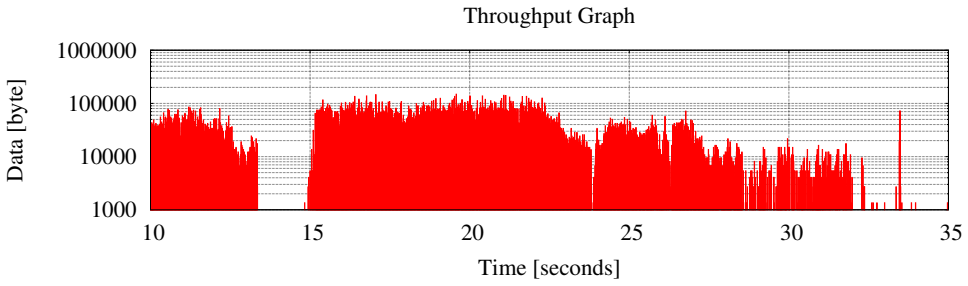

Figure 4.3: TCP Throughput graph using ZTE MF823 with Netcom with a sample rate of 10 milliseconds.

4.1.2 After Handover

Each of the three handover examples referenced in this subsection corresponds to the three figures shown in the subsection above. Figure 4.4 is the throughput graph after the handover to figure 4.1 and 4.5 corresponds to 4.2 and 4.6 to 4.3 .

A completed handover means that the modem has changed technology from one to another, and is ready to receive or send data packets. This subsection will show the effects of a handover on active connections by observing the throughput graphs of three selected tests.

In figure 4.4 , the client starts to receive the data packets at 27 seconds. The total length of the handover was 5 seconds. Three seconds after the client starts to receive data packets, the throughput decreases. There is also a noticeable spike in the throughput. This will be explained later in this chapter.

The graph in figure 4.5 shows a similar increase in throughput once the handover has completed. This lasts only for less than a second and is followed by a normalization of the throughput level. This graph is the post handover test shown in figure 4.2 which stopped receiving packets after 15 seconds. Once the handover was complete, the client starts to receive packets after 18 seconds, leaving 3 seconds of stalled connection time.

In the last of these throughput graphs, figure 4.6, the connection is more bursty and sporadic. There is no clear throughput increase, as seen in the previous two connections.

In this graph, it should be observed that the client received the last packets after 36 seconds, before the handover, and starts to receive packets 47 seconds after test start. The handover takes 7 seconds, but does not start until 3 seconds after the last packet was received.

4.1.3 Summary

These figures show how a connection behaves as it is entering, and finishing a handover. It can affect the connection quite differently, depending on the factors involved. These results will be further studied with more in depth analysis to better understand what happens during a handover.

An example of a worst-case handover is displayed in figure 4.7 . This was captured using the ZTE MF821D modem on a bus with Telenor. This was the slowest handover combined with the highest stalled connection time observed in a test. The modem used 28 seconds to change technology from LTE to HSPA+. Despite this, it took 68 more seconds before the client and server started to exchange data packets.

An interesting observation is that the server sends a packet to the client after 58 seconds from the start (in the figure mentioned above) and the client responds to it. However, this does not trigger a resumption of the connection.

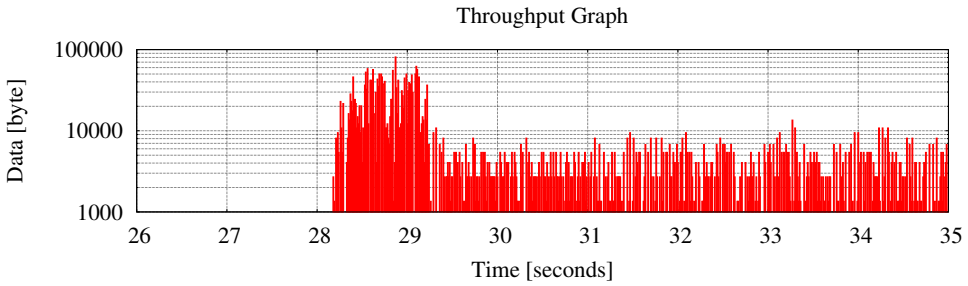

Figure 4.4: Post handover throughput graph with Netcom using ZTE MF823. Sample rate of 10 milliseconds.

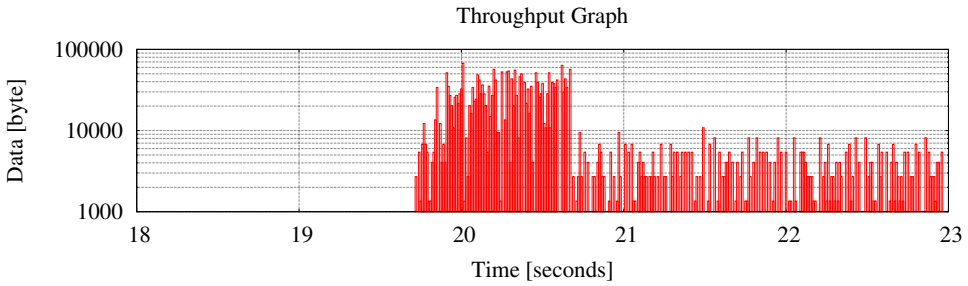

Figure 4.5: Post handover throughput graph with Netcom using ZTE MF823. Sample rate of 10 milliseconds.

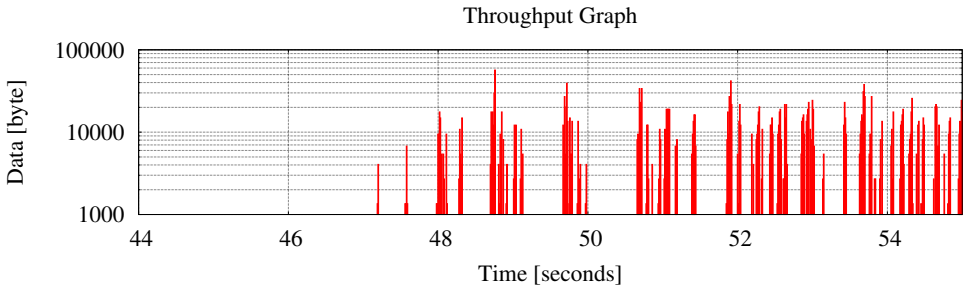

Figure 4.6: Post handover throughput graph with Netcom using ZTE MF823. Sample rate of 10 milliseconds.

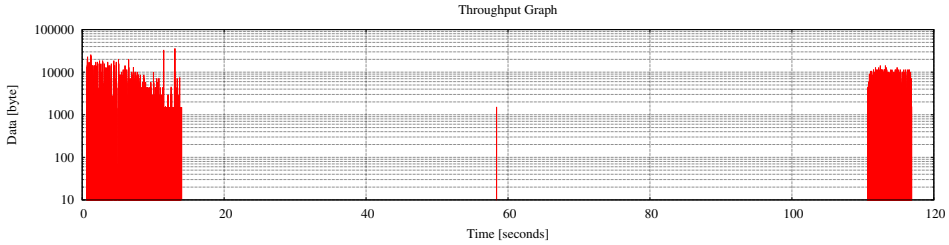

Figure 4.7: Throughput graph using Telenor with ZTE MF821D. Sample rate of 10 milliseconds.

4.1.4 Latency during handover

Before a handover, packet latency will increase as the signal becomes weaker. This can be observed in figure 4.8, 4.9 and 4.10 which show parts of the packet exchange flow for the three previous mentioned figures.

In these figures a complete handover is observed and it is possible to observe the latency increase on packets as the signal is getting worse. The red lines represent the data packets sent from the server, while the turquoise lines represent the corresponding ACK packets.

While not every data packet has a corresponding ACK packet, this is due to duplicate ACKs which are not displayed in these graphs.

In both figure 4.8 and 4.9 ACK latency is seen increasing before the handover.

In these figures latency increases for both data packets and ACK packets and it shows how the connection becomes more bursty as the signal is faltering.

4.2 Comparison

This section will compare the throughput flow of TCP connections with UDP both before and after a handover. This is done in order to answer the questions posed in the UDP section in chapter 3.

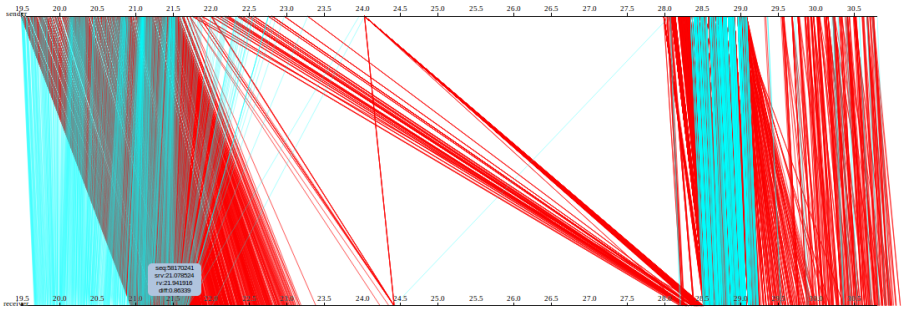

Figure 4.8: PACKet graph shows a part of the communication between the server and client before and after the handover for figures 4.1 and 4.4

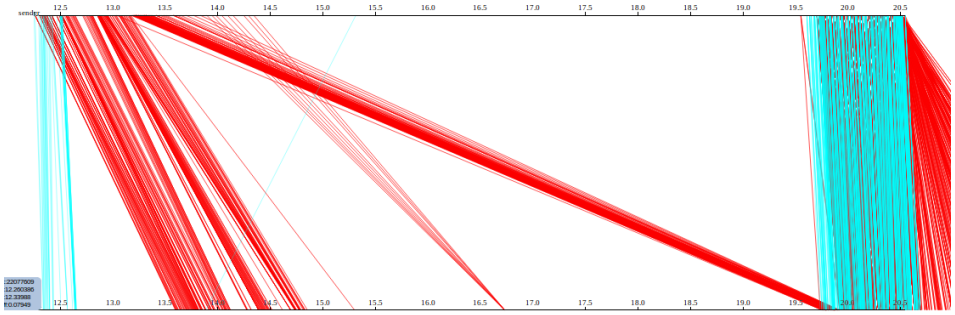

Figure 4.9: PACKet graph shows a part of the communication between the server and client before and after the handover for figures 4.2 and 4.5

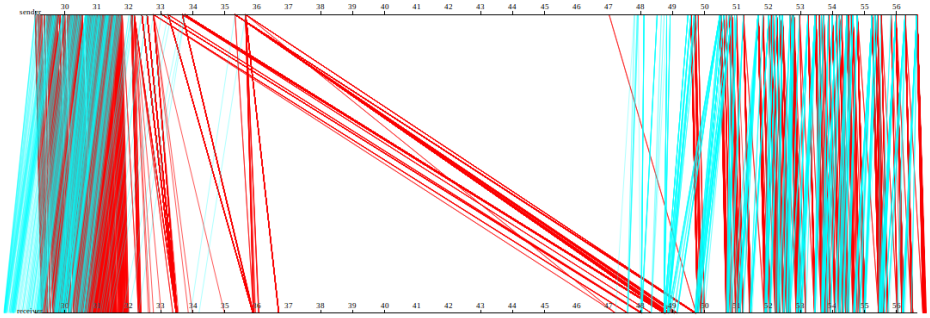

Figure 4.10: PACKet graph shows a part of the communication between the server and client before and after the handover for figures 4.3 and 4.6

4.2.1 TCP vs UDP

In order to determine if there is room for improvement in the way a handover affects TCP, a series of UDP tests have been conducted with the main goal of finding a potential optimum for packet exchange after a handover has been completed. The tests used the bandwidth-generator program described in chapter 3 .

Before any comparison can be done, we must first present some UDP figures. In figure 4.11 we can see a throughput graph showing a UDP test done with Telenor using the ZTE MF823 modem. In this test, the reported time when the modem is between technologies (When the handover is occurring) is reported in table 4.13 under test #2 as 5 seconds.

In the mentioned throughput figure, the receiver starts to receive packets after 4 seconds, at this time the modem states that it isn't connected to any technology. This can be explained by the fact that the tool used to retrieve modem statistics is updated only once a second. This means that in this test, the client started to receive data instantly.

Another test can be seen in figure 4.12 . During this test it took the modem 31 seconds to complete the handover. Once the handover was completed, the client starts to receive data instantly. In this test, there were two occurrences where the client suddenly stopped receiving data. The handover goes from LTE to UMTS, which is the largest gap in the connection. This is followed by two subsequently smaller gaps in which the modem reports that it changed technology from UMTS to HSPA+ (Extended 3G) followed by a change from HSPA+ to UMTS before going back to HSPA+. Exactly why this pattern occurs is not know, but it will be briefly discussed in chapter 5 on page 70.

After the handover has completed and the client has started to receive data packets in the figure mentioned above the throughput is noticeably lower by several orders of magnitude. During this time the modem states that it is using UMTS followed by a change to HSPA+. These intermissions post-handover where the modem changes technology from HSPA+ to UMTS and back again has been observed in several of the UDP tests, but has not occurred in any of the TCP tests.

All the other UDP tests show that the client receives data packets once the handover has completed.

There are several differences between TCP and UDP. The main difference is the lack of a stalling in UDP tests. During UDP testing there has not been any occurrences of a connection stalling which has been observed with several TCP tests.

UDP does however show intermissions in which is is unable to receive data post-handover, which has not been observed in TCP testing.

4.2.2 Handover Time

The length of a handover appears to be dependent on both the network, and the hardware. The term stalling in relation to a

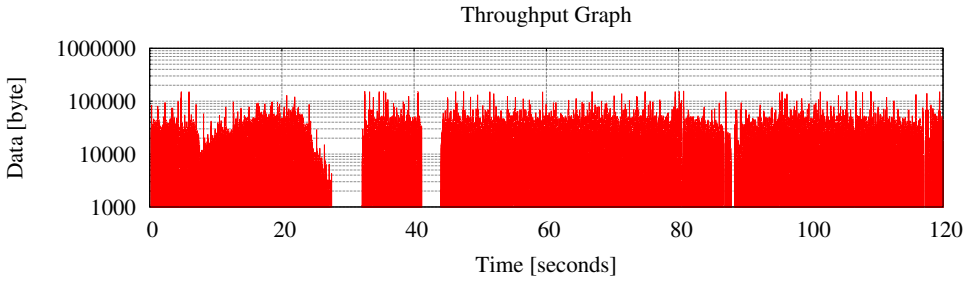

Figure 4.11: UDP throughput with Telenor, using ZTE MF823 graph with a sample rate of 10 milliseconds.

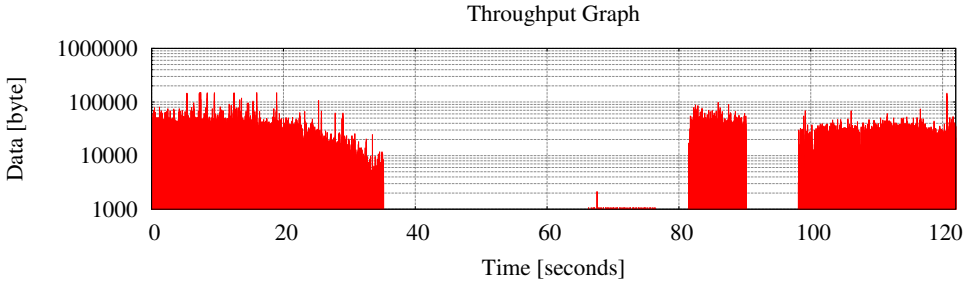

Figure 4.12: UDP throughput with Netcom, using ZTE MF823 graph with a sample rate of 10 milliseconds.

Test #	**Netcom**	**Telenor**
1	10	12
2	4	5
3	31	4
4	3	4
5	12	4
Average	**12**	**5.8**

Figure 4.13: UDP handover length in seconds, using ZTE MF823

Test #	Netcom	Telenor
1	19	6
2	4	6
3	3	4
4	10	5
5	4	4
Average	**8**	**5**

Figure 4.14: TCP handover length in seconds, using ZTE MF823

connection is used when the handover has completed, but neither the receiver nor sender is exchanging packets. This means that if the handover takes 10 seconds, and the connection starts 5 seconds later, leaving a total time of no packet exchange 15 seconds, it has a stalled connection time of 5 seconds.

In table 4.15 is a list showing the length of a handover for one set of tests using modem ZTE MF821D, and it shows the stalling time of the connection. In the first test, the handover takes 28 seconds from start to finish, it means that for 28 seconds the modem was not connected to any technology. After those 28 seconds, it took 43 more seconds before the server transmits a packet back to the client, leaving the connection stalling for a total of 43 seconds.

This can be observed in the other tables as well, showing that the modem ZTE MF823 performed a far quicker handover.

In the tables displayed the length of the handover as reported by the modem is displayed along with the length of the stalled connection and the total time it took for the connection to resume from the start of the handover.

During the tests, the difference in handover length time between Netcom and Telenor became apparent. Telenor appears to struggle more with a handover when using ZTE MF821D, while Netcom was a bit slower than Telenor when using ZTE MF823.

4.3 Analysis

This section will show an analysis of what happens at a lower level, it will show an analysis of the TCP connection and give a more detailed description of what happens before and after a handover for both the client and server.

Test #	Handover time	Stalled connection time
1	28	43
2	28	29
3	30	30
4	26	12
5	28	29
6	28	26
7	28	25
8	28	1
9	28	68
10	26	15
Average	**27.8**	**27.8**

Figure 4.15: TCP handover length and stalling connection time in seconds, using ZTE MF821D with Telenor

4.3.1 TCP

When a client goes through a handover, it will be unable to receive and send data packets to the server. From the perspective of the server the client is unresponsive. This can lead the server to retransmit data packets as they start to time out, depending on the length of the handover.

Once the client responds the server will observe a high RTT and will generally assume congestion in the network. This will result in the server responding accordingly and start to prevent congestion while the client might be in a non-congested and fully operational network.

The following sub sections will explain what happens on both the client side and server side as this handover occurs.

Client

Before the handover has occurred, the signal will degrade and become weaker. This can lead to a lower throughput rate and an increased risk of packet loss. In a few of the tests done, the client did not experience packet loss prior to the handover.

In the events of a packet loss the client receives packets out-of-order and will start to transmit duplicate ACK packets. Packet loss before entering the handover has been observed in a majority of the test cases.

After the handover has been completed, the client will be ready to receive data packets, and will be waiting for the server to start sending

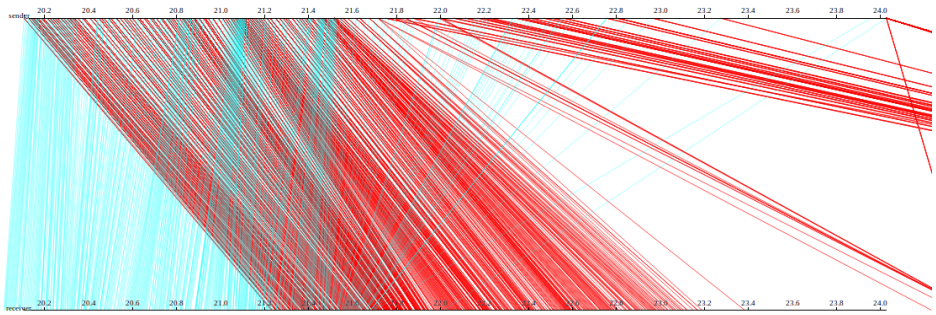

Figure 4.16: Close up of figure 4.8 before the handover.

data packets.

Two possibilities arises after a handover. Either the client receives packets from the network, which was in flight prior to the handover ended, or it receives nothing. When it receives data these are usually out-of-order or older packets which it has acknowledged prior to the handover. These will result in duplicate ACK packets and the server will respond accordingly.

If it does not receive any data packets it will wait.

Server

As the client is losing signal and prior to the initiation of a handover it will experience both packet delay and packet loss. As the signal is decreasing ACK packets will be received with an increasing latency and then they will be dropped. This increase in delay can be observed in figure 4.16.

As the ACK packets are starting to drop, the server will continue to send out data packets until it has filled the sending window. Once the send window is full the server will wait for ACK packets from the client, but the client is currently preforming a handover.

This will lead to a stall in the connection, with the server waiting for ACK packets while the client is changing technology. As seen in section 4.2.2 the length of a handover can vary and is unpredictable. This can lead to a potential deadlock broken by the packet timer.

Depending on the length of the handover, the server will experience several packet retransmission timeouts (RTO) and start to retransmit the data packet. It will then reset the timer and double the timer length. This will continue until the client has completed the handover in which there are two cases:

1. The client replies with an ACK after the handover.

2. The server experiences packet retransmission timeout. (RTO)

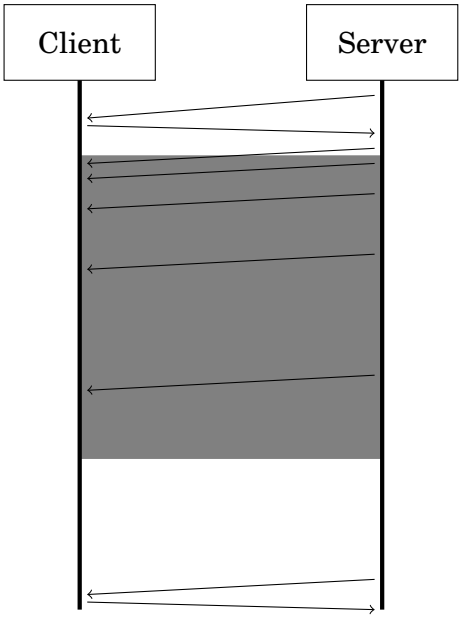

Figure 4.17: A representation of a handover affected by an exponential backoff. The grey area represents the handover.

In the first case the client will have received packets in flight, and will send an ACK (or several) to the server, which will lead the server to respond and the connection should continue.

In the second case the server will wait until the timer times out again and then retransmit the timed out packet. As the client has completed the handover it will respond and the connection will continue.

The second case is the cause of the stalling of a connection, and the length of the stalling depends on the handover length. The longer the handover takes, the longer the timer grows. An example of how the timer can cause a stalling is shown in figure 4.17 . This figure shows an example where the client does not receive any data packets after the handover, a stalling occurs.

Summary

When the client has completed a handover there are potential issues. The client can experience packet loss and a stalled connection.

This stalling is a result of the handover and depends on the length of a handover. A slow handover can result in a longer stalling, but does not necessary imply one as the client can complete the handover just prior to the next timeout.

As seen this results in two potential triggers for packet exchange between the client and server.

1. The client receives packets after the handover has completed.

2. A packet times out at the server which is then retransmitted.

In the first case, the client will usually (In every case this happened) receive several packets. This always resulted in out-of-order packets and in turn resulted in several dupACK packets being sent back to the server.

If the server receives more than 3 duplicate ACK packets, it will assume packet loss and enter ”Recovery” state.

Example:
In the figures 4.1 and 4.4 the client triggers the resumption of the connection when it receives a series of buffered packets from the network which it then ACKs. These packets were transmitted just before the client began the handover and because of packet loss prior to the handover, it ends up receiving a buffer containing out-of-order packets which resulted in the client transmitting up to 300 dupACKs back to the server. This results in the server entering a 'Recovery' state.

In the second case, there are two possible sub-cases:

1. The packet it receives is an old, acknowledge packet and the acknowledgement packet was dropped.

2. It receives a new unacknowledged packet.

It is not possible for the client to receive an out-of-order packet when the server retransmits a timed out packet. The packet that times out will be the next packet after the last acknowledged packet. Meaning the packet with a sequence number equal to the acknowledgement number in the last ACK packet, which is the next packet the client expects.

The first case was observed in every case. This is because ACK packets are dropped prior to the client entering the handover.

The second case has not been observed in any of the tests, but it is still a possible, albeit low-probable, case. In this case, there should be no issues once the handover has completed beyond the result of an RTO (Retransmission Timeout, described in chapter 2). This can only be the case when all packets sent, including the one that times out, was dropped.

In the first case this will usually result in at least one duplicate ACK. The tests show that when the sender receives this duplicate ACK it will send new data to the server, which will result in several new duplicate ACKs followed by the server entering ”Recovery” state.

The second case was not observed, but should result in the server entering the ”Loss” state as described in 2.4. This will lead to the server marking every packet sent as lost and they will be retransmitted.

In both cases the server will enter ”Recovery” state and collapse the congestion window. Based on the tests, there does not appear to be any way a handover will not result in the server entering either ”Recovery”, ”Disorder” or ”Loss” state with one notable exception, which will be further discussed in section 4.3.3 . This exception is how Netcom handles a handover, but is limited to HTTP over port 80. Their solution is to use

a proxy that lies between the client and server.

These cases poses a question:

- Is there potential for improvement in the TCP protocol before and after a handover?

This question can be broken down into several questions:

1. Is it necessary to enter the ”Recovery” state once a connection has resumed?
2. Is it possible to avoid the server from entering ”Recovery” state after a handover?
3. Is it possible to prevent stalling if the handover takes too long?

These questions will be discussed further in the next chapter.

4.3.2 Throughput growth after handover

After the handover has completed, a few questions can be asked about the connections throughput.

1. Is there a difference in the throughput growth between TCP and UDP?
2. Is there a difference in the throughput growth between Netcom and Telenor?

Is there a difference in throughput growth between TCP and UDP?

In the UDP tests, the server sends a constant amount of data. Thus the throughput growth for a UDP test will be limited to the modem and carrier network and not with the protocol it self. This will be useful when comparing how fast a TCP connections throughput grows after a handover.

If the growth of these differ, it could mean that there is some issue with how TCP handles a growth after the handover.

One potential issue when comparing TCP and UDP is that UDP has shown to have issues with intermittent pauses in the connection, where it changes technology between HSPA+ and UMTS. However, it might still be possible to compare the throughput growth. Unless there is an apparent issues with the growth itself, then it should still be possible to compare the two.

In figure 4.18 and figure 4.19 are two post-handover throughput plots for UDP tests with Netcom and Telenor (respectively) using ZTE MF823 running the bandwidth generator program.

Tables 4.1 through 4.4 contains the throughput of a series of tests using both UDP and TCP. Each table contains the average throughput

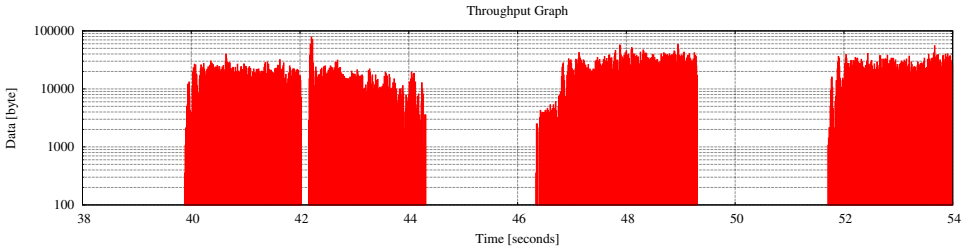

Figure 4.18: UDP throughput growth using ZTE MF823 with Netcom as carrier. Sample rate is 10 milliseconds.

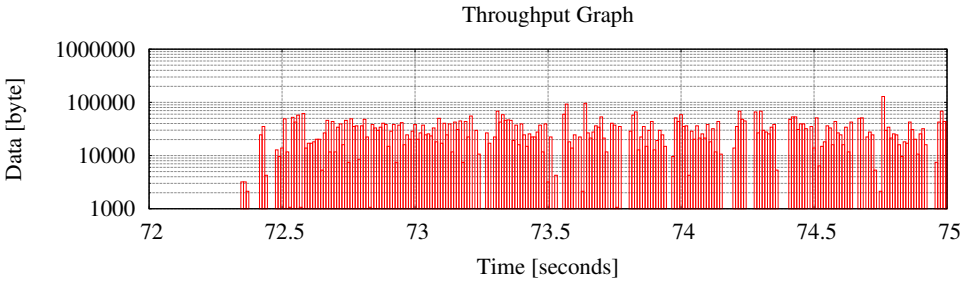

Figure 4.19: UDP throughput growth using ZTE MF823 with Telenor as carrier. Sample rate is 10 milliseconds.

Test #	**Average**
1	18.5 Mbit/s
2	15.7 Mbit/s
3	42.6 Kbit/s - 21.3 Mbit/s
4	20.6 Mbit/s
5	19.7 Mbit/s
Average	**14.9 Mbit/s - 19.16 Mbit/s**

Table 4.1: Average throughput bandwidth during UDP test with Netcom one second after the handover has completed.

Test #	**Average**
1	2.9 Mbit/s
2	17.2 Mbit/s
3	18.1 Mbit/s
4	0.75 Mbit/s
5	17.19 Mbit/s
Average	**11.2 Mbit/s**

Table 4.2: Average throughput bandwidth during TCP test with Netcom one second average after the connection has resumed.

one second after they first receive data packets. For UDP tests this will be right after the handover, while for TCP this depends on when the connection is restarted after the handover. In some cases this happened fast, others this happened after a few seconds of stalling.

By comparing these values it should be possible to compare the growth and try to determine if one is more efficient than the other.

In table 4.1 the average throughput for 5 UDP tests, for the duration of one second after the handover, is listed. These tests were done using the bandwidth generator with 20 MBit/s output. Note that test #3 has two values. The first one denote the first second after the handover, at this time the modem was using UMTS and experienced low bandwidth until it changed to HSPA+. The second value is the throughput after the change in technology.

Table 4.3 shows the TCP test using PHCC. These tests show the throughput one second after the client first received a data packet, and not one second after the handover had completed.

The average throughput for all the UDP tests is 19.16 Mbit/s

Test #	Average
1	5.4 Mbit/s
2	13.19 Mbit/s
3	20.7 Mbit/s
4	20.4 Mbit/s
5	19 Mbit/s
Average	**15.7 Mbit/s**

Table 4.3: Average throughput bandwidth during UDP test with Telenor one second after the handover has completed.

Test #	Average
1	9.3 Mbit/s
2	15 Mbit/s
3	3.3 Mbit/s
4	20.6 Mbit/s
5	15.8 Mbit/s
Average	**12.8 Mbit/s**

Table 4.4: Average throughput bandwidth during TCP test with Telenor one second average after the connection has resumed.

while the TCP shows 11.2 Mbit/s. While there aren't necessarily enough test to get a statistical valid average comparison, this might show a tendency towards UDP. In section 4.1 several post handover throughput graphs where displayed showing a tendency for Netcom to have a higher throughput post-handover, lasting a few seconds followed a decline and normalization.

The average throughput for UDP in Telenor tests is 15.7 Mbit/s, while for TCP it is 12.8 Mbit/s. This again might show a tendency towards a better throughput for UDP, however there are limitations as to what can be achieved with TCP. This will be discussed further in section 5.3.4.

While this isn't conclusive, UDP does appear to have a fast throughput rate post-handover.

Is there a difference in throughput growth between Netcom and Telenor

By comparing the post-handover throughput plots of the following figures, a difference between Netcom and Telenor becomes more

Test #	Netcom	Telenor
1	2.23 Mbit/s	18.81 Mbit/s
2	2.19 Mbit/s	19.95 Mbit/s
3	2.21 Mbit/s	10.47 Mbit/s
4	2.34 Mbit/s	19.33 Mbit/s
5	1.9 Mbit/s	15.5 Mbit/s
Average	2.17 Mbit/s	16.8 Mbit/s

Table 4.5: Average throughput bandwidth for the duration of a connection post handover for Netcom and Telenor.

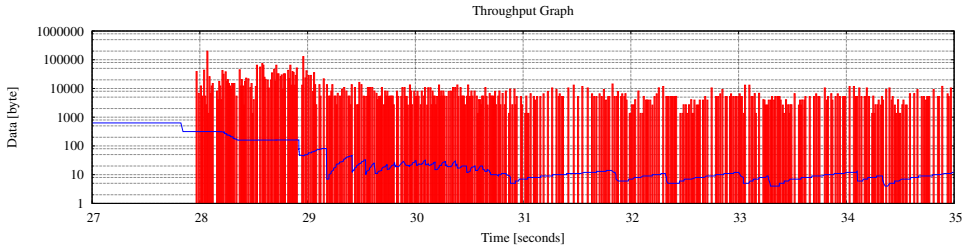

Figure 4.20: Throughput graph of a Netcom test, showing post handover with a sample rate of 10 milliseconds.

apparent. These figures are the result of a driving test using ZTE MF823 modem and the PHCC program running on port 8080.

The tables above display the throughput of a few tests showing that the average throughput post handover between Netcom and Telenor does not differentiate by much the first second, however as the connection continues Netcom sees a decline in throughput and achieves an overall lower throughput average after a few seconds on the 3G network. Telenor does not show these results and maintains a higher throughput level than Netcom on the 3G network.

Table 4.5 shows the difference in throughput through out the rest of the connection, once the modem has connected to 3G for both Netcom and Telenor.

To better understand these, an analysis of two tests post-handover and what happens at the server will follow.

Figure 4.20 shows the post handover throughput for the server with an overlay showing the congestion window. This figure shows the throughput graph from the servers side for test #2 which is the same figures as seen in figure 4.1 and 4.4 .

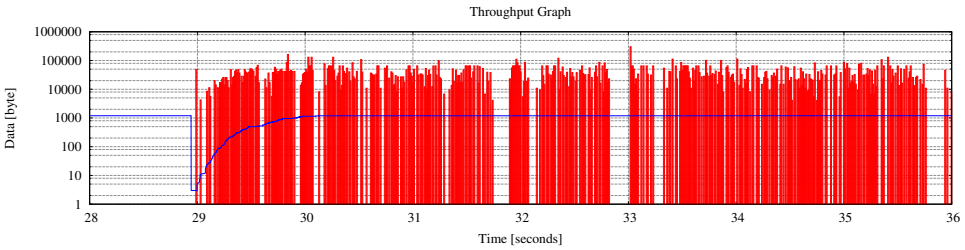

Figure 4.21: Throughput graph of a Telenor test, showing post handover with a sample rate of 10 milliseconds.

Prior to the loss detection the congestion window is set to 632, this is halved once the server detects packet loss by receiving several duplicate ACK packets. This results in the server changing the internal TCP state from ”Open” to ”Recovery”. The server starts to decrease the congestion window for each incoming ACK packet.

Once the packet loss has been repaired the server changes to ”Open” and the congestion window starts to grow slightly before congestion is detected. The server continues to use Congestion Avoidance throughout the remaining time of the connection as congestion is detected.

This was common for all the tests done with Netcom that day.

The second figure shows the server throughput for test #2 of the Telenor TCP tests. Here the server will first enter the ”Disorder” state followed by ”Open” and quickly enter the ”Loss” state. During this state the congestion window is reset and all outstanding packets are marked as lost. When in the ”Lost” state, the server can only return to ”Open” state once all the lost packets have been successfully acknowledged.

During this time the congestion window is increasing for each incoming ACK packet, and the server has repaired the connection at around 29.9 seconds after the start of the test. At which points it leaves the ”Loss” state and enters the ”Open” state.

This was common for the tests done with Telenor that day.

While there appears to be a difference between Netcom and Telenor, this seems to be mostly limited to the throughput level post handover. This could be attributed to several factors which will be discussed in section 5.1 in the next chapter. However the growth of the throughput appears to be similar for either network and is most likely more dependent on the congestion control algorithm, rather than the network itself.

4.3.3 Netcoms Proxy Solution

This sub section presents Netcom’s proxy solution, which has been observed when running tests with HTTP over port 80.

How the proxy works

Netcom utilizes a proxy solution to handle TCP connections for HTTP over port 80 that appears to be quite effective in preventing congestion and packet loss during a handover.

The proxy is observable from the server-side dump when compared to the client-side dump as the communication between proxy and server has window scaling disabled, allowing the proxy to control the receiving window size (Window scaling is described in 2.3.1 in chapter 2 on page 13). This window size is updated by the proxy, but the communication between client and proxy uses automatic window scaling.

When the proxy is used, it takes over communication between the client and server. It lies between the two as a hidden proxy, controlling the TCP connection. While this is hidden, it is apparent when one compares the network dump for the client-side communication and the server-side communication.

From the observations, it appears as if the proxy separates the communication between the client and server completely. The proxy will ACK and receive packets which it will then transmit to the client. This is an assumption made as results show that the client can experience packet loss between the client and proxy, while there has not been any observed packet loss between the proxy and server.

When the client is changing technologies, the proxy will send a Zero-Window packet to the server to block communication. This could be done for several reasons. While the proxy does appear to buffer the packets before sending them to the client, blocking could be used in order to prevent the buffer from filling up and causing packet loss when the proxy is unable to "offload" packets by sending them to the client.

The proxy and server will exchange keep-alive packet when the receiving window is zero, in order to keep the connection alive. Once the client is able to receive data packets, the proxy will update the receiving window and the connection will continue.

It will also block communication with the server when the client has lost coverage. Some tests show that when the modem has low LTE signal (Zero bars) the proxy would send Zero-Window packets to the server, however this did not result in a handover as the signal became stronger.

An example of the latency flow between client, server and proxy is seen in 4.22 .

This does appear to have a good effect on the connection flow. It should also be noted that Telenor does not employ any such proxy.

How the proxy effects a connection

When the proxy service is used, it will block the connection when the client is either experiencing low signal or entering a handover. This will prevent packet loss and prevent stalling.

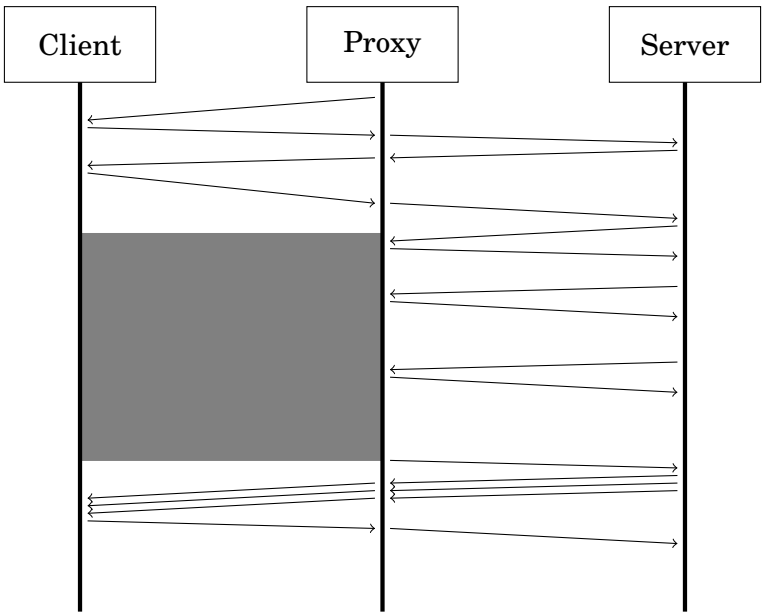

Figure 4.22: A representation of how the proxy prevents the server from sending packets to the client during a handover. The grey area represents the handover.

The proxy will maintain the connection to the server when the client is experiencing either weak signal or changing technologies.

Figure 4.23 and 4.24 shows a real life case with the proxy. The first figure is the client side throughput and the second shows the server side. In these cases it is possible to see the proxy in action during the handover.

One thing to notice is the throughput difference. The proxy will split up the packets, sending smaller packets than it receives from the server.

There are two gaps in both plots. The first gap is the result of a falling signal, but one that did not result in a handover. The signal strength increased. The second gap represents the handover from 4G to 3G. During both gaps the proxy blocks the connection by sending Zero-Window packets to the server while the client was unable to receive data. These packets are visible in the server-side dump, and can be seen as the tiny packets slightly below the 100 mark.

After the handover, the proxy resumes the connection and starts sending data to the client. However, in this test the proxy ends up sending the client packets out of order and had to start repairing the packet loss, which is seen in the graph as the increased throughput.

Since the proxy controls the receiving window size, which can be seen in the server throughput, the server will fill up the receiving window and wait for window size updates from the proxy to. This can be seen as the tiny pauses, or tiny gaps in the connection.

The client will also experience these similar gaps, which is the client waiting for data from the proxy. The tests show that the proxy can wait

1-20 milliseconds between packets, which will show up in the graph as tiny gaps.

The proxy appears to add around 500 ms latency, and also increases the throughput post handover. Exactly why this increase happens is not known. The packet latency can be seen in figure 4.25 .

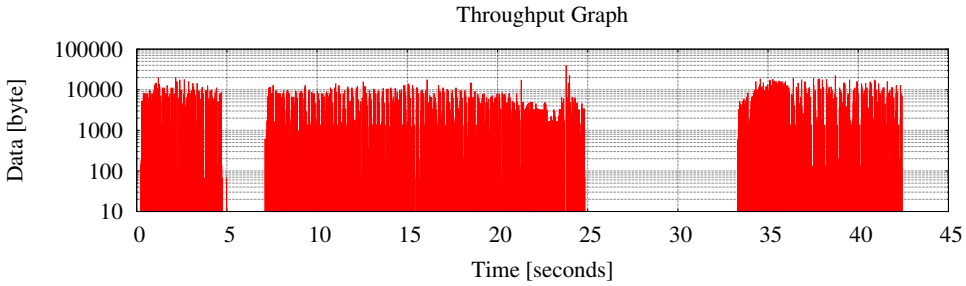

Figure 4.23: Client throughput graph with a sample rate of 10 milliseconds.

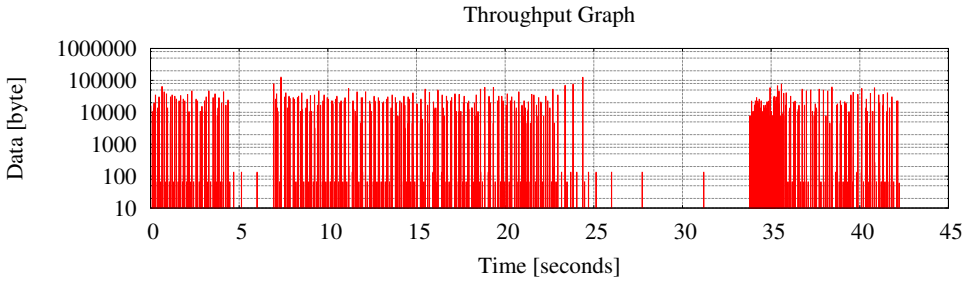

Figure 4.24: Server throughput graph with a sample rate of 10 milliseconds.

The two following figures shows the first 5 seconds post handover for both client and server. Figure 4.26 shows the throughput after the gap in 4.23 , while 4.27 shows the gap in 4.24 .

Once the client starts to receive data from the proxy, it experiences packet loss and the proxy has to repair this. The repair takes less than a second, and the connection is fixed before 34 seconds.

The server does not detect any packet loss, as this is limited to the client and proxy only. It continues to send new data packets to the proxy.

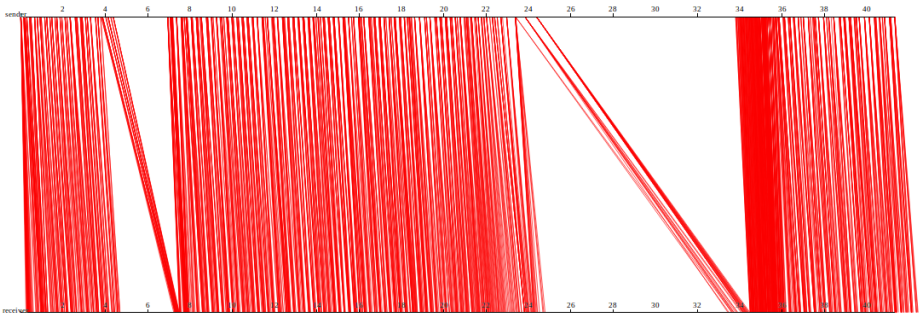

Figure 4.25: A graph showing when a packet was sent and received. Top represents server and bottom the client.

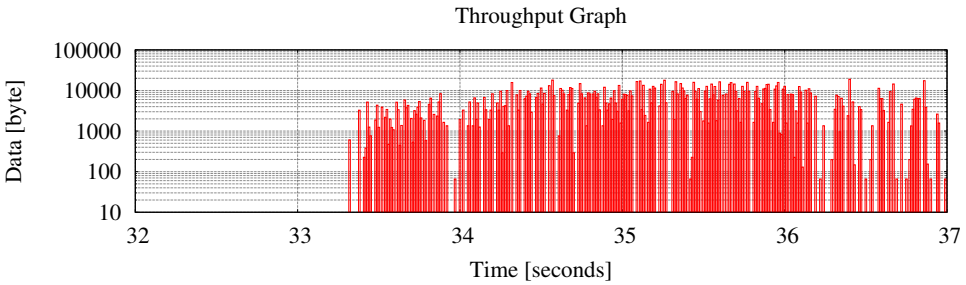

Figure 4.26: Client throughput graph with a sample rate of 10 millisecond.

Summary

The proxy appears to work quite well, it prevents the server from experiencing any packet loss as a result of the handover and provides a smooth transition for the client.

It works well, and by controlling the receiving window it allows for control over the throughput for the client.

4.4 Summary

This chapter has presented the results and analysis of what happens during a handover, both before and after and how it affects an active TCP connection. Also presented Netcom's proxy solution for HTTP browsing. Through the analysis of the results, several problems have been outlined. In the next chapter a discussion around these problems and potential solutions are proposed.

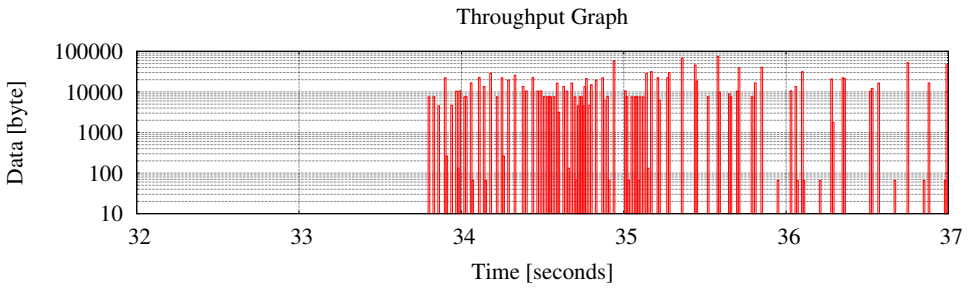

Figure 4.27: Server throughput graph with a sample rate of 10 milliseconds.

Part III

Conclusion

Chapter 5

Discussion

In the previous chapter, results were presented followed by an analysis of what happens. This chapter will focus on a discussion around a few of the problems posed and suggestions for future work.

5.1 Questions Posed

In chapter 4 several questions have been posed, and this section will discuss these and provide some insight.

5.1.1 TCP

In section 4.3.1 several questions were asked regarding how TCP behaves post handover. To reiterate these questions:

1. Is it necessary to enter the ”Recovery” state once a connection has resumed?

2. Is it possible to avoid the server from entering ”Recovery” state after a handover?

3. Is it possible to prevent stalling if the handover takes too long?

Is it necessary to enter the ”Recovery” state once a connection has resumed?

The testing have shown that as a receiver will experience packet loss either prior or after a handover. While it is possible for the receiver not to experience any packet loss, this has not been observed.

As the sender needs to repair the connection when packet loss has occurred, it is necessary to enter either the ”Recovery” state. However, this is only necessary when packet loss is detected.

This means that to prevent the sender from entering ”Recovery” state, packet loss must be prevented.

Is it possible to avoid the server from entering ”Recovery” state after a handover?

As stated in the previous question, preventing the ”Recovery” state would require preventing packet loss. While preventing packet loss in general is not possible, whether or not it is possible to prevent packet loss as a result of a handover is another question.

As was seen in section 4.3.3 Netcom uses a proxy solution which lies between the client and server. While there was packet loss, this was limited to the client and proxy, the server did not observe any packet loss as a result of the handover and otherwise poor signal quality.

This method prevents the server from entering ”Recovery” state as a result of the handover, and could be a method of avoiding this.

The proxy appears to be an effective method of preventing packet loss between the server and client.

Is it possible to prevent stalling if the handover takes too long?

The source of the stalling was shown to be the packet timer, or more specifically how the timer grows. As the timer grows exponentially the potential for stalling will occur depending on the length of the handover.

A potential solution would be to prevent the timer from growing exponentially, or by adding a mechanism that would allow the client to trigger the connection to resume, even if it has not received any packets.

While adding a mechanism for the client to trigger the connection would solve the problem, it also requires more additions to the protocol. Modifying the timer growth would be far more simpler and could potentially prevent stalling beyond at least one second.

Netcom’s proxy solution also prevents this, as the network knows when the client is able to receive data packets again. This method works well to prevent stalling of a connection.

Suggestions for different methods of preventing stalling can be found under Future Work.

5.2 Thesis Limitations

This section will go into the limitations in this thesis. This will cover the limitations on the tests, and the results from the testing.

5.2.1 TCP Tests

There are several limitations in the TCP tests that should be expanded upon before accepting any of the results. The number of tests done is not significant enough to determine the average throughput for either network, and there has not been enough handovers captured to estimate a proper average handover length.

While the number of tests were not significant enough to say much about the average handover time, nor sufficient in variety when it comes

to modems. The goal of this thesis was to see if there was any potential problems caused by a handover. As the results show, there are several issues which should be focused on.

The testing of TCP has showed that the length it takes to complete a handover can affect the stalling of a connection. When the client is changing technology, the server will usually experience packet timeouts. As seen, these timeouts can grow excessively and will potentially affect a connection.

There are also other limitations which are as follows:

- Congestion Control Algorithm
- High throughput

Congestion control Algorithm

The server was set to use the congestion control algorithm New Reno, which has an improved retransmission. This might result in a more optimized throughput growth rate when compared to other congestion control algorithms.

In order to understand how this limitation affects the tests, a better understanding of how different congestion control algorithms affect throughput growth is needed.

High throughput testing

These tests had a high bandwidth focus where the server would send 20 Mbit/s to the user. This amount of data is unrealistic for an average users bandwidth usage. This is not realistic for a user roaming the network, and further testing should be required.

High bandwidth was used as it would give a better understanding of how the throughput growth works on mobile broadband after a handover has completed.

Although testing with low throughput should be done, if would most likely not yield any different results as TCP should behave the same before and after a handover.

5.2.2 Lack of solution testing

This thesis did not attempt to solve any problems or issues, it was limited in scope and has as a goal to study how TCP behaves during a handover.

Several potential solutions are proposed and future work is required in an attempt to improve any problems discovered.

5.2.3 Modem limitations

There were only two modems used for testing, which can be a limiting factor as hardware appears to be an important issue with handovers.

There was a clear difference in handover length for both modems and stalling was only a severe issue with ZTE MF821D.

5.2.4 Network limitation

The networks tested were only Netcom and Telenor. Unlike Telenor, Netcom had a fast handover with ZTE MF821D which could point to a difference in how the networks are configured in regards to a handover.

This could mean that other networks maintain other configurations and thus has a difference in handover behaviour.

5.3 Problems

This sections will break down the problems observed and discuss each problem in more detail, in order to fully understand the issues at hand before presenting any possible solution.

5.3.1 Handover Time

The time it takes for a client to complete a handover will directly affect how long it takes for the connection to resume.

As seen in table 4.15 , the length a handover takes can cause the connection to stall for quite some time. If a user is experiencing multiple handovers while travelling, this problem will prevent a user from effectively utilizing several different types of online services.

An example list of a few services affected can be seen in table 5.1 with a description as to how a handover would affect the service.

There might be several reasons for why the handover is slow and in this can be attributed to several factors. As only one of the modems used in this thesis showed a significantly slower handover, a few potential reasons is outlined below.

- Hardware
- Software
- Network

Hardware

Tables 4.14 and 4.15 show a list of the time it takes each of the modems to complete a handover. It shows that modems have a significant impact on the handover length and is most likely tied into the physical hardware specifications. How this would affect other devices has not been tested, but should yield interesting results.

Service	Description
Online video streaming	Online video streaming services such as YouTube, Netflix and Twitch stream videos to a user. These videoes are usually buffered, however, with the limitations of mobile broadband, the extent of the buffering is limited to the available bandwidth a user has. If a user is experiencing a slow handover with a large stalling, it might take up to a minute before he can receive more data for buffering.
Online music streaming	Similar to the video streaming, music streaming services such as Spotify, Google Music and WiMP are also potentially affected. These services do provide an offline feature to listen to music without streaming them, they are usually limited to premium users. Users without this feature will also potentially be affected by a slow handover.

Table 5.1: Different services which could be directly affected by a slow handover.

Software

It could be possible that the qmi-dialer software used to initiate and handle the ZTE MF821D modem is the cause for the delayed handover. If this is the case then it should affect both networks similarly, however, this has not been the case. Netcom handovers using this modem has been quite fast and it was only Telenor that suffered massive delays.

However, this could be further tested by using the same modem with a different set up. If there is another way of extracting the relevant statistical and technological data from this modem, it should be tested on both Windows and OS X in order to ensure that this was not the qmi-dialer.

It should also be possible to do a ”blind” test, if it is not possible to retrieve modem data on other operating systems. This would require server-side network dumps, which could be used to determine a potential stalling.

Network

How the network handles a handover could also explain the difference in the connections. This could simply be a difference in network configuration or how a handover is done.

If this was the case, then it should be reflected in the handover length of the other modem, but this is doubtful as the handover length of the other modem was quite fast for both Netcom and Telenor.

5.3.2 UDP Intermissions

As described in sub section 4.3 in chapter 4 on page 46 , UDP tests show an oddity, at seemingly random times the client stops receiving data as the modem changes technology for no apparent reason. This could also pose issues with online services, like VOIP, Skype or others that rely on UDP communication, rather than TCP.

However, these intermissions could also be the result of the following issues, which the tests do not cover:

- Modem hardware
- Bandwidth limitations

The modem hardware could be a possible explanation for this, as there were only two modems used in testing. In order to ensure that it is not the fault of modems, several other modems should be used during testing in order to explain this issue.

Bandwidth limitations could also be a potential explanation. These tests were done with a high throughput level to see how quickly a UDP test would gain maximum throughput after a handover. In order to

determine that this is not the cause several tests with lower bandwidth should be done. However, the TCP tests have also been high bandwidth and the issue at hand has not occurred during these.

5.3.3 Connection stalling

An important observation has been the stalling of a connection. In the previous chapter it was shown that stalling is caused by the packet timers growth function. As this grows exponentially, the probability for a stalled connection increases.

There was several tests that showed the importance of hardware involved. ZTE MF821D showed a slower handover mechanism than ZTE MF823.

5.3.4 Throughput Growth

In the previous chapter a comparison between the throughput growth of TCP and UDP was shown. This showed that UDP maintained a higher throughput than TCP once the client and server started to exchange data packets.

With the UDP testing the client was simply receiving data packets, and was not responding, it should be able to maintain a higher throughput rate than TCP. As TCP has a more significant overhead because of the need to maintain flow control, congestion control and to ensure that the receiver acknowledges the packets.

There are several factors that can also affect the throughput growth, and throughput in general these are outlined below.

Network capacity

Testing was done in the middle of the day, which can affect the throughput. If there were more users connected to Netcoms network at the time of testing, it could explain why Netcom had a lower overall throughput.

As testing the network capacity was not the goal of this thesis, these values should not necessarily reflect on the network and the throughput values given in this thesis are merely indications.

Packet loss

Because UDP does not know about packet loss, and the client only was a passive receiver. The increased throughput can be attributed to packet loss in UDP, as it that does not require the server to retransmit any lost packets. The server was sending 20 Mbit/s and by looking at the table 4.3 and 4.1 the client receives close to this.

TCP would have to repair any packet loss once the client and server starts to exchange packets.

5.4 Further Work

This section will propose future studies which will cover further testing, as well as suggestions for future work involving methods to prevent stalling.

5.4.1 Handover Frequency

As this thesis has been limited to observing how a handover affects a TCP connection, it would be useful to do a study where the goal is to determine how frequently a user experiences a handover. Such a study would reveal more details about how a handover affects TCP.

More in depth studies about the handover frequency and handover time would also give more data regarding how often the stalling of a connection affects users. While 4G is currently being deployed, there are still users without 4G capable devices so a study that observes the handover frequency between several generations would be useful.

5.4.2 Multiple devices

Testing with several devices such as smartphones would be useful in order to determine how widespread these issues are.

As more people are using smartphones with 3G and 4G, understanding how a smartphone handles a handover seems imperative.

5.4.3 Proxy Emulation

It might be possible to emulate the proxy in some ways, by having the client send a Zero Window packet when the signal is failing. This would require modifications to the kernel to expose key functions, as well as require external input (Modem signal) or heuristics in order to detect packet loss and throughput decline as a result of poor signal.

While this suggestion might have some benefits, it would seem improbable and impractical to use, as the client would still be unable to respond to keep-alive packets.

5.4.4 Modifying Exponential Backoff

Linear timeout has been integrated into the Linux Kernel, but it is limited to Thin TCP Streams [37]. When Thin TCP streams are enabled, it will activate a linear timeout when there are less than 4 packets in flight. Linear timeout will be attempted 6 times, and if the client is still not responding it will fall back to exponential backoff.

Since the exponential backoff algorithm can cause a stall in the connection, using linear timeouts make sense in this context.

By using the same system as thin TCP streams, with a linear timeout method prior to an exponential backoff, it should be possible to prevent an extended stalling connection. This should be further tested, but would potentially aid in the handover.

While this would not necessarily prevent stalling, it would potentially cut the stalling time.

5.4.5 Removal of Exponential Backoff

An alternative to modify the exponential backoff would be to remove it completely. This has been previously suggestion in paper "Removing Exponential Backoff from TCP" [38] which studies the effect of removing exponential backoff from TCP.

This would be an interesting test, as it should in theory enable the server and receiver to start communicating earlier, thus preventing a long stalling of the connection.

This would require a big change to the TCP protocol and should be scrutinized heavily. However testing with this approach should yield some interesting results.

5.4.6 Congestion Control Algorithms

The TCP tests in this thesis have only used Reno as a congestion control algorithm. It would be useful to do more testing using several different types of congestion control algorithms in order to see how each would affect a user.

In chapter 2 several congestion control algorithms were named with a short description. By testing several of these it would be possible to determine if there is an optimal congestion control algorithm that would allow a quicker throughput growth after a handover has been complete.

5.5 Summary

This chapter discussed some of the results and limitations in this thesis. It also presents new areas of future work and suggestions for further testing.

Chapter 6

Conclusion

6.1 Summary

This thesis has studied how an active TCP connection over mobile broadband is affected when the modem changes technology from 4G to 3G. The goal has been to understand what happens and how this affects a TCP connection with an attempt of discovering any underlying issues with the protocol as a result of a handover.

In order to get a better understanding of how a handover affects TCP, several tests have been conducted using multiple networks and modems testing both TCP and UDP. The main focus has been on what occurs after a handover has completed.

UDP tests were conducted in order to compare throughput growth with TCP as UDP has a lower overhead. Results from the UDP tests showed that the client was able to receive data instantly after a handover, while TCP results were mixed. Some tests showed an instant packet exchange while others suffered from stalling in the connection.

The stalling observed was the result of a packet timer that grows exponentially and can cause delay up to 60 seconds. Theoretically this can grow up to 120 seconds when the server runs Linux.

These tests also revealed that UDP has a higher throughput growth than TCP. This makes sense considering the overhead involved with TCP.

During this thesis it was discovered that Netcom uses a proxy solution to prevent packet loss between the client and server, but this is limited to HTTP over port 80. The proxy will block the connection when the user is experiencing low signal quality or the modem is preforming a handover. This solution will also prevent stalling, which can be a problem.

Several suggestions for areas that need more study has been proposed along with a few suggestions to prevent stalling. These areas involve further network study, more modem testing with different devices and different congestion control algorithms in order to compare how these will affect TCP after a handover. The suggestions for preventing stalling mostly involve modifying how TCP's packet timer

works.

While this thesis had several limitations, the work done has shown that there areas with TCP that can be improved upon, and while more work is required to get a better understanding of the impacts.

Bibliography

[1] Statista. *Number of smartphones sold to end users worldwide from 2007 to 2013 (in million units)*. URL: http://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/ (visited on 18/01/2015).

[2] http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2014/ITU_Key_2005-2014_ICT_data.xls.

[3] Kathi Ann Brown. *Bringing Information to People: Celebrating the Wireless Decade*. 1993. URL: http://www.milestonespast.com/exbringing.htm (visited on 17/01/2015).

[4] Elisa. *History*. URL: http://corporate.elisa.com/on-elisa/history/ (visited on 17/01/2015).

[5] Karin Jansson. *First in the world with 4G*. URL: http://www.teliasonerahistory.com/pioneering-the-future/pioneering-the-future/first-in-the-world-with-4g/ (visited on 17/01/2015).

[6] GSM History. *GSM History*. URL: http://www.gsmhistory.com/the-beginnings/ (visited on 03/01/2015).

[7] NTT DoCoMo. *A Brand New Mobile Millennium Ericsson/C-ATT/DoCoMo jointly demonstrate pioneering W-CDMA technology at PT/Wireless*. 1999. URL: https://web.archive.org/web/20130311204138/http://www.nttdocomo.com/pr/1999/001070.html (visited on 18/01/2015).

[8] *HSDPA (High-Speed Downlink Packet Access)*. URL: http://www.gsmarena.com/glossary.php3?term=hsdpa.

[9] Ian Poole. *3G HSUPA - High Speed Uplink Packet Access*. URL: http://www.radio-electronics.com/info/cellulartelecomms/3g-hspa/hsupa-high-speed-uplink-packet-access.php.

[10] GSMA. *HSPA*. URL: http://www.gsma.com/aboutus/gsm-technology/hspa.

[11] *LTE Encyclopedia*. URL: https://sites.google.com/site/lteencyclopedia/home.

[12] Yogen Dalal Vinton Cerf and Carl Sunshine. *SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM*. RFC 675. Dec. 1974, pp. 1–70. URL: http://tools.ietf.org/html/rfc675.

[13] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. 5th. Pearson, 2011. ISBN: 0132553171, 9780132553179.

[14] K. Ramakrishnan et al. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. Sept. 2001. URL: http://tools.ietf.org/html/rfc3168.

[15] N. Spring et al. *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*. RFC 3540. June 2003. URL: http://tools.ietf.org/html/rfc3540.

[16] Scil100 Sergiodc2 Marty Pauley. *TCP state diagram*. 2010. URL: http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg.

[17] Clemente. *TCP CLOSE*. 2011. URL: http://en.wikipedia.org/wiki/File:TCP_CLOSE.svg.

[18] Alexander Krivács Schrøder Wikimedia Commons. *Sliding Window with a 2-bit sequence and a size of 1*. File: `Sliding_Window.svg`. 2008. URL: http://en.wikipedia.org/wiki/File:Sliding_Window.svg.

[19] V. Mathis et al. *TCP Selective Acknowledgment Options*. RFC 2018. Oct. 1996. URL: https://tools.ietf.org/html/rfc2018.

[20] E. Blanton M. Allman V. Paxson. *TCP Congestion Control*. RFC 5681. Sept. 2009. URL: http://tools.ietf.org/html/rfc5681.

[21] T. Henderson et al. *The NewReno Modification to TCP's Fast Recovery Algorithm*. RFC 6582. Apr. 2012. URL: http://tools.ietf.org/html/rfc6582.

[22] *BIC and CUBIC*. URL: http://research.csc.ncsu.edu/netsrv/?q=content/bic-and-cubic (visited on 18/01/2015).

[23] V. Paxson et al. *Computing TCP's Retransmission Timer*. RFC 6298. June 2011. URL: http://tools.ietf.org/html/rfc6298.

[24] Pasi Sarolahti and Alexey Kuznetsov. 'Congestion Control in Linux TCP'. In: *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2002, pp. 49–62. ISBN: 1-880446-01-4. URL: http://dl.acm.org/citation.cfm?id=647056.715932.

[25] Jon Postel. *User Datagram Protocol (RFC 768)*. IETF Request For Comments. Aug. 1980.

[26] J. Postel. *Internet Protocol*. RFC 791. Sept. 1981. URL: http://tools.ietf.org/html/rfc791.

[27] *4G USB DATA MODEM - USER MANUAL*. http://www.zte.com.au/downloads/manuals/MF821_Manual.pdf. 2012.

[28] *4G USB DATA MODEM - USER MANUAL*. http://www.zte.com.au/downloads/manuals/MF823_Help.pdf. 2013.

[29] Tcpdump team. *TCPDump*. http://www.tcpdump.org/.

[30] Gerald Combs et al. *Wireshark*. https://www.wireshark.org/.

[31] Kristian Evensen. *Multi Client*. https://github.com/kristrev/multi. 2013.

[32] Kristian Evensen. *QMI-Dailer*. https://github.com/kristrev/qmi-dialer. 2013.

[33] Kristian Evensen. *Bandwidth Estimator*. https://github.com/kristrev/bandwidth-estimator. 2012.

[34] Linux Foundation. *TCP Probe*. http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe.

[35] Thomas Williams, Colin Kelley and many others. *Gnuplot 4.4: an interactive plotting program*. http://www.gnuplot.info/. Mar. 2010.

[36] *GNU Wget*. https://www.gnu.org/software/wget/.

[37] http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=36e31b0af58728071e8023cf8e20c5166b700717.

[38] Amit Mondal and Aleksandar Kuzmanovic. 'Removing Exponential Backoff from TCP'. In: *SIGCOMM Comput. Commun. Rev.* 38.5 (Sept. 2008), pp. 17–28. ISSN: 0146-4833. DOI: 10.1145/1452335.1452338. URL: http://doi.acm.org/10.1145/1452335.1452338.

Appendices

Appendix A

Appendix

PHCC

This is a very basic tool that sends as much data as possible from the server to the client. When the client stops receiving TCP data it will start pinging with UDP packets. These packets contain a sequence number which increases for each ping. It will attempt to ping the server once every 500 ms.

Both the client and server can be found at the following git repository: https://bitbucket.org/patrisk/phcc

TCP Probe

This is based on an extended version of TCP probe, which can be found here: https://bitbucket.org/bendikro/tcp_probe_rdb The modifications done for this thesis as been to add port numbers for each stream in the output line and include the internal TCP congestion state.

Changes can be found in the git repository: https://bitbucket.org/patrisk/tcp_probe_rdb